VC Verification IP
UART
HDL User Guide

Version O-2018.09, September 2018

Contents

Preface

About This Guide

This guide contains installation, setup, and usage material for HDL users of the VC VIP for UART, and for design or verification engineers who want to verify UART operations using a HDL testbench written in Verilog. Readers are assumed to be familiar with UART and Verilog.

Guide Organization

The chapters of this guide are organized as follows:

Chapter 1,“Introduction”, introduces the UART VIP and its features.

Chapter 2, “Installation and Setup”, describes system requirements and provides instructions on how to install, configure, and begin using the UART VIP.

Chapter 3, “General Concepts”, introduces the UART VIP within a HDL environment and describes the data objects and components that comprise the VIP.

Chapter 4, “Usage Notes”, covers the scenarios which require specific use model in the UART VIP.

Chapter A, “Reporting Problems”, outlines the process for working through and reporting UART VIP issues.

Web Resources

- Documentation through SolvNet: https://solvnet.synopsys.com (Synopsys password required)
- Synopsys Common Licensing (SCL): http://www.synopsys.com/keys

Customer Support

For Customer Support, perform any of the following tasks:

- Enter a call through SolvNet:
  - Go to http://solvnet.synopsys.com/EnterACall and click **Open A Support Case** to enter a call.
  - Provide the requested information, including:
    - **Product**: Verification IP
    - **Sub Product 1**: UART SVT
    - **Product Version**: O-2018.09
    - Fill in the remaining fields according to your environment and your issue
- Send an e-mail message to support_center@synopsys.com
    - Include Product name, Sub-Product name, and Product Version as discussed previously.
- Telephone your local support center:
    - North America:

      Call 1-800-245-8005 from 7 AM to 5:30 PM Pacific time, Monday through Friday
    - All other countries:

      http://www.synopsys.com/Support/GlobalSupportCenters

1 Introduction

The UART VIP supports the verification of SOC designs that include interfaces implementing UART specifications. This document describes the use of the VIP in testbenches that comply with Verilog. The SVT Verilog VIP provides the following features:

- Protocol functionality and abstraction
- Error injection
- Protocol checks
- Rapid creation of complex tests
- Proven testbench architecture that provides maximum reuse, scalability, and modularity
- Proven verification approach
- Self-checking tests

This chapter consists of the following sub-sections:

- "Product Overview" on page 7
- "Components" on page 8
- "Simulator Support" on page 9
- "UART Feature Support" on page 9

This document assumes that you are familiar with UART and Verilog.

1.1 Product Overview

The UART HDL VIP leverages advanced verification techniques in creating a versatile testbench environment. It provides Bus Functional Model (BFM), checker, and monitor modules, which can be instantiated in a Verilog testbench. The BFM modules support the functionality of the creation of transactions. The checker modules provide the functionality of checking and reporting the protocol correctness. The monitor modules provide transaction logging. The monitor and checker combined together are independent modules and you can use them with any BFM. They are both enabled, by default.

1.2 Components

This section discusses the following components of the UART HDL VIP:

- BFM
- Checker
- Monitor

1.2.1 BFM

The BFM module implements the behavioral model of the specification, which is used to test the Design Under Test (DUT). Tables 1-1 shows the tasks available in the module.

Table 1-1 BFM Tasks

BFM Tasks	Description
do_cmd()	Instructs the BFM module to generate desired transactions on the bus.
do_cfg()	Configures the BFM module with legal parameters. You can use it to change environment parameters.
do_cfg_rd()	Reads the value of BFM parameters.
do_err()	Configures the BFM module to generate errors.
do_pkt()	Generates data packets for transmission.

1.2.2 Checker

The checker module performs specification checks on the traffic generated between the DUT and BFM. The checker performs the real-time reporting of errors on the clock in which the end symbol of a packet is received. It displays error messages if a violation occurs.

The checker module contains the following task:

- do_cfg(): It configures the checker module.

1.2.3 Monitor

The monitor module acts as a passive component that snoops on the traffic between the DUT and the BFM module. The monitor module generates traffic logs in both the transmit and receive direction.

The monitor module contains the following task:

- do_cfg(): It configures the monitor module.

1.3 Simulator Support

The UART VIP suite supports the following language and simulator:

- Language: Verilog
- Simulator: VCS, NC Sim, and ModelSim

1.4 UART Feature Support

The UART VIP suite supports the following protocol-related features:

- Capable to verify Data Communication Equipment (DCE) and Data Terminal Equipment (DTE) DUT.
- Full duplex operation, that is, simultaneous transmission and reception.
- Programmable baud generator, which divides any input clock by 1 to (2*16-1) and generates the 16x clock.
- Configurable Baud Rate Divisor, that is, you can select the baud rate divisor according to the UART DUT requirement.
- Configurable Fractional Baud Rate Divisor, that is, you can select the fractional baud rate divisor according to the UART DUT requirement.
- Out-of-band flow control.
- Configurable data framing with 9 bits as well as 5 bits, 6 bits, 7 bits, and 8 bits.
- Programmable 1 stop bit and 2 stop bits.
- Line-break generation and detection.
- Programmable parity (Even, odd, stick, or no-parity).
- Receiver FIFO.
- Simple and flexible BFM tasks:
  - do_cmd() for commands and parameters that change with every command. The do_cmd provides the following command feature:
    - Number of data bytes involved in current transactions.
  - do_cfg() for configuring BFM parameters with the following features:
    - Configures the BFM for the buffer size of the receiver.
    - Configures the BFM for the break condition when the BFM transmits all zero packets.
    - Configures the BFM for the delay time of the Request to Send (RTS) assertion.
    - Configures the BFM for the delay time of the Clear to Send (CTS) assertion.
    - Configures the BFM for an inter-cycle delay time.
  - do_cfg_rd() to read BFM parameters.
- do_pkt() to pass data to BFMs.
- do_err() to inject errors with the following features:
  - Occurrence of invalid parity
  - Occurrence of invalid stop bits
  - Occurrence of the break condition

- Exhaustive built-in tests: The UART VIP allows you to carry out the following in-built tests:
  - Basic tests: Consists of tests designed to test the basic functionality of the UART-based design.
  - Error tests: Consists of test cases to inject errors in transactions.
  - Random tests: Consists of exhaustive random tests designed to test the combination of parameters and commands.
  - User tests: Consists of tests designed for UART feature completeness like baud divisor,data-parity, fractional dividor tests etc.
- Programmable UART monitor to debug complex-test scenarios:
  - Performance analysis
  - Simulation summary
- Programmable UART checker:
  - Exhaustive checks
  - Individual rules (enabled or disabled)
- Programmable message-logging capabilities: The UART VIP allows you to specify the following message levels:
  - Error, warning, log, debug, or info
- A UART agent, which can be instantiated in the users' testbench environment. For more information, see "General Concepts" on page 25.

2 Installation and Setup

This chapter leads you through the installing and setting up the UART VIP. After completing this checklist, the provided example testbench will be operational and the UART VIP will be ready to use.

This chapter consists of the following major steps:

- "Verifying Hardware Requirements" on page 11
- "Verifying Software Requirements" on page 11
- "Preparing for Installation" on page 12
- "Downloading and Installing" on page 13
- "Setting Up a Testbench Design Directory" on page 15
- "Licensing Information" on page 16
- "Environment Variable and Path Settings" on page 17
- "Determining Your Model Version" on page 18
- "Integrating a UART Verification IP into Your Testbench" on page 18

2.1 Verifying Hardware Requirements

The UART VIP requires the following configuration for Solaris or Linux workstation:

- 400 MB available disk space for installation
- 16 GB Virtual memory (recommended)
- FTP anonymous access to ftp.synopsys.com (optional)

2.2 Verifying Software Requirements

The UART VIP is qualified for use with the certain versions of platforms and simulators. This section lists the software required by the UART VIP and consists of the following sub-sections:

- "Platform/OS and Simulator Software" on page 12
- "SCL Software" on page 12

- "Other Third-Party Software" on page 12

2.2.1 Platform/OS and Simulator Software

Platform/OS and VCS: You need the versions of your platform/OS and simulator that have been qualified for use.

For more details, refer UART VIP Release Notes.

2.2.2 SCL Software

The Synopsys Common Licensing (SCL) software provides the licensing function for the UART VIP. For details on acquiring the SCL software, see the installation instructions in "Licensing Information" on page 16.

2.2.3 Other Third-Party Software

Following is the list of other third party software:

- **Adobe Acrobat**: The UART VIP documents are available in Acrobat PDF files. Adobe Acrobat Reader is available for free from http://www.adobe.com.
- **HTML Browser:** The UART VIP includes a class-reference documentation in HTML that supports the following browser or platform combinations:
  - Microsoft Internet Explorer 6.0 or later (Windows)
  - Firefox 1.0 or later (Windows and Linux)
  - Netscape 7.x (Windows and Linux)

2.3 Preparing for Installation

Perform the following steps to prepare for installation:

a. Set `DESIGNWARE_HOME` to the absolute path where the UART VIP is to be installed:

```
setenv DESIGNWARE_HOME absolute_path_to_designware_home
```

b. Ensure that your environment and PATH variables are set correctly, including the following:

- `DESIGNWARE_HOME/bin` – The absolute path as described in the previous step.
- `LM_LICENSE_FILE` – The absolute path to a file that contains the license keys for your third-party tools. Also, include the absolute path to the third-party executable in your PATH variable.

```
% setenv LM_LICENSE_FILE <my_license_file|port@host>
```

- `SNPSLMD_LICENSE_FILE` – The absolute path to a file that contains the license keys for the SCL software or the `port@host` reference to this file.

```
% setenv SNPSLMD_LICENSE_FILE $LM_LICENSE_FILE
<my_Synopsys_license_file|port@host>
```

- `DW_LICENSE_FILE` – The absolute path to a file that contains the license keys for VIP product software or the port@host reference to this file.

```
% setenv DW_LICENSE_FILE <my_VIP_license_file|port@host>
```

2.4 Downloading and Installing

You can download software from the Download center using either HTTPS or FTP, or with a command-line FTP session. If you do not know your Synopsys SolvNet password or you do not remember it, go to http://solvnet.synopsys.com.

You require the passive mode of FTP. The passive command toggles between the passive and active mode. If your FTP utility does not support the passive mode, use HTTP. For additional information, refer to the following web page:

https://www.synopsys.com/apps/protected/support/EST-FTP_Accelerator_Help_Page.html

Attention The Electronic Software Transfer (EST) system only displays products that your site is entitled to download. If the product you are looking for is not available, contact est-ext@synopsys.com.

This section consists of the following sub-sections:

- "Downloading From EST (Download Center)" on page 13
- "Downloading Using FTP With a Web Browser" on page 14

2.4.1 Downloading From EST (Download Center)

a. Point your web browser to http://solvnet.synopsys.com.

b. Enter your Synopsys SolvNet Username and Password.

c. Click the `Sign In` button.

d. Make the following selections on SolvNet to download the `.run` file of the VIP (See Figure 2-1).

   i. `Downloads` tab

   ii. VC VIP Library product releases

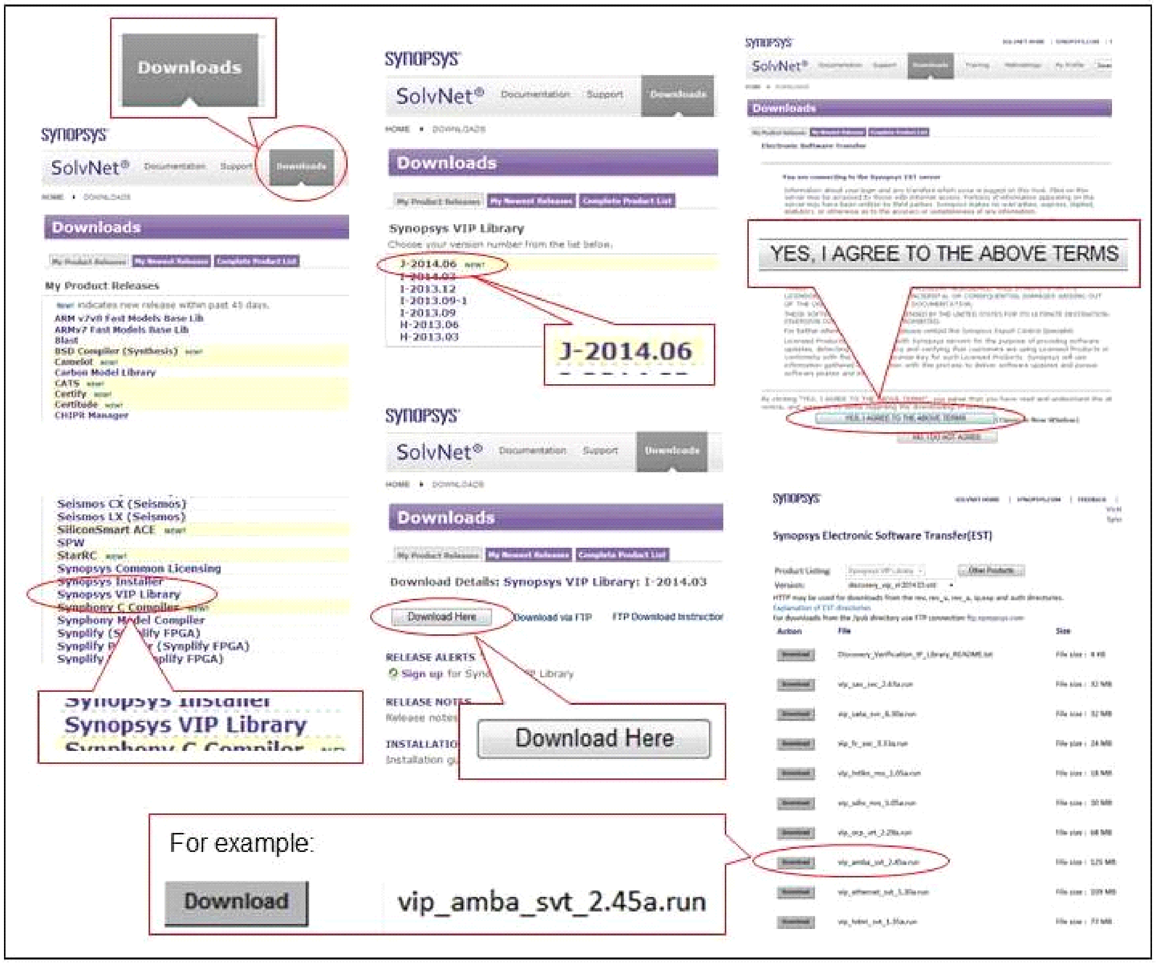
iii. <release_version>

iv. `Download Here` button

v. `Yes, I Agree to the Above Terms` button

vi. Download `.run` file for the VIP

Figure 2-1 SolvNet Selections for VIP Download



e. Set the `DESIGNWARE_HOME` environment variable to a path where you want to install the VIP.

```
% setenv DESIGNWARE_HOME VIP_installation_path
```

f. Execute the `.run` file by invoking its filename. The VIP is unpacked and all files and directories are installed under the path specified by the `DESIGNWARE_HOME` environment variable. The `.run` file can be executed from any directory. The important step is to set the `DESIGNWARE_HOME` environment variable before executing the `.run` file.

2.4.2 Downloading Using FTP With a Web Browser

Follow Step a to Step e of Section 2.4.1 and then perform the following steps:

a. Click the **Download via FTP** link instead of the **Download Here** button.

b. Click the **Click Here To Download** button.

c. Select the file(s) that you want to download.

d. Follow browser prompts to select a destination location.

2.5 Setting Up a Testbench Design Directory

A design directory is where UART VIP is set up for use in a testbench. The `dw_vip_setup` utility is provided as design directory is required for using VIP.

The `dw_vip_setup` utility allows you to create the design directory (`design_dir`), which contains the VIP components, support files (include files), and examples (if any). Add a specific version of UART VIP from `DESIGNWARE_HOME` to a design directory.

For a complete description of `dw_vip_setup`, see "The dw_vip_setup Utility" on page 19.

The following models are provided with UART VIP:

- `uart_agent_svt`
- `uart_txrx_svt`
- `uart_monitor_svt`

Use the following command to create a design directory and add a model to be used in a testbench:

```
%$DESIGNWARE_HOME/bin/dw_vip_setup -path design_dir -a(dd) <model1> -svtb
For example, %$DESIGNWARE_HOME/bin/dw_vip_setup -path design_dir -a uart_agent_svt -svtb
Or

%$DESIGNWARE_HOME/bin/dw_vip_setup -path design_dir -a(dd) <model1> <model2>
<model3> -svtb
For example, %$DESIGNWARE_HOME/bin/dw_vip_setup -path design_dir -a uart_agent_svt
uart_txrx_svt uart_monitor_svt -svtb
Or

%$DESIGNWARE_HOME/bin/dw_vip_setup -path design_dir -a(dd) -model_list
<input_file_containing_models_one_per_line>  -svtb
For example, %$DESIGNWARE_HOME/bin/dw_vip_setup -path design_dir -a -model_list
<filelist> -svtb
cat filelist:
uart_agent_svt
uart_txrx_svt
uart_monitor_svt
```

After running the above command, the model files are installed at the following location:

```
<design_dir>/include and <design_dir>/src
```

Note: You need to specify the pointer to these installed directories on Simulator analyze or compile options.

2.6 Licensing Information

UART VIP uses the Synopsys Common Licensing (SCL) software to control its usage. You can find general SCL information from the following link,

http://www.synopsys.com/keys

You can enable UART VIP by performing the license check in the order listed below. Once a required feature or a set of features are successfully checked-out, the VIP stops looking for other licenses. The type of license depends on product and portfolio type. You can use one of the following set of hierarchies:

- `VIP-UART-SVT`
- `VIP-PROTOCOL-SVT`
- `VIP-SOC-LIBRARY-SVT`
- `VIP-LIBRARY-SVT + DesignWare-Regression`

Only one license is consumed per simulation session, irrespective of how many VIP products are instantiated in the design.

The licensing key must reside in the files that are indicated by specific environment variables. For information about setting these licensing environment variables, see "Environment Variable and Path Settings" on page 17.

This section consists of the following sub-sections:

- "Controlling License Usage" on page 16
- "License Polling" on page 17
- "Simulation License Suspension" on page 17
- "Environment Variable and Path Settings" on page 17

2.6.1 Controlling License Usage

You can control which license is used, using the `DW_LICENSE_OVERRIDE` environment variable, as follows:

- To use only the `DESIGNWARE-REGRESSION` and `VIP-LIBRARY-SVT` licenses, set `DW_LICENSE_OVERRIDE` to `DESIGNWARE-REGRESSION` and `VIP-LIBRARY-SVT`.

- To use only the `VIP-UART-SVT` license, set `DW_LICENSE_OVERRIDE` to `VIP-UART-SVT`. If `DW_LICENSE_OVERRIDE` is set to a value and the corresponding feature is not available, a license error message is issued.

Note This release supports only the suite license features.

2.6.2 License Polling

If you request a license and none are available, License Polling allows your request to exist until a license is available instead of exiting immediately. To control License Polling, use the `DW_WAIT_LICENSE` environment variable in the following way:

- To enable License Polling, set the `DW_WAIT_LICENSE` environment variable to 1.
- To disable License Polling, unset the `DW_WAIT_LICENSE` environment variable. By default, license polling is disabled.

2.6.3 Simulation License Suspension

All the Verification IP products support License Suspension. The simulators, which support License suspension, allow the model to check-in its license token while the simulator is suspended and then checkout the license token when the simulation is resumed.

Note This capability is simulator-specific; all simulators do not support license check-in during License Suspension.

2.7 Environment Variable and Path Settings

The following environment variables and path settings are required by the UART VIP verification models:

1. Set `DESIGNWARE_HOME` to the following absolute path where the Synopsys UART VIP is recommended to be installed:
   `setenv DESIGNWARE_HOME absolute_path_to_designware_home`
2. Ensure that your environment and `PATH` variables are set correctly:

- `DESIGNWARE_HOME`: The absolute path to where the VIP is installed.
- `DW_LICENSE_FILE`: The absolute path to file that contains the license keys for the VIP product software or the port@host reference to this file.
- `SNPSLMD_LICENSE_FILE`: The absolute path to file(s) that contains the license keys for Synopsys software (VIP and/or other Synopsys Software tools) or the port@host reference to this file.

Note
For faster license checkout of Synopsys VIP software, ensure to place the desired license files at the front of the list of arguments to SNPSLMD_LICENSE_FILE.

- `LM_LICENSE_FILE`: The absolute path to a file that contains the license keys for both Synopsys software and/or your third-party tools.

Note
You can set the Synopsys VIP License using any of the three license variables in the following order:

```
DW_LICENSE_FILE -> SNPSLMD_LICENSE_FILE -> LM_LICENSE_FILE
```

- If DW_LICENSE_FILE environment variable is enabled, the VIP will ignore SNPSLMD_LICENSE_FILE and LM_LICENSE_FILE settings. Therefore, to get the most efficient Synopsys VIP license checkout performance, set the DW_LICENSE_FILE with only the License servers which contain Synopsys VIP licenses. Also, include the absolute path to the third party executable in your PATH variable.

Simulator-Specific Settings

Your simulation environment and `PATH` variables must be set as required to support your simulator.

2.8 Determining Your Model Version

The following steps describe how to check your model version:

Note Verification IP products are released and versioned by the suite and not by the individual model. The version number of a model indicates the suite version.

- To determine the versions of VIP models installed in your `$DESIGNWARE_HOME` tree, use the following setup utility:

```
% $DESIGNWARE_HOME/bin/dw_vip_setup -i home
```

- To determine the versions of VIP models in your design directory, use the following setup utility:

```
% $DESIGNWARE_HOME/bin/dw_vip_setup -p design_dir_path -i design
```

2.9 Integrating a UART Verification IP into Your Testbench

After installing the VIP, use the following procedures to set up the VIP for use in testbenches:

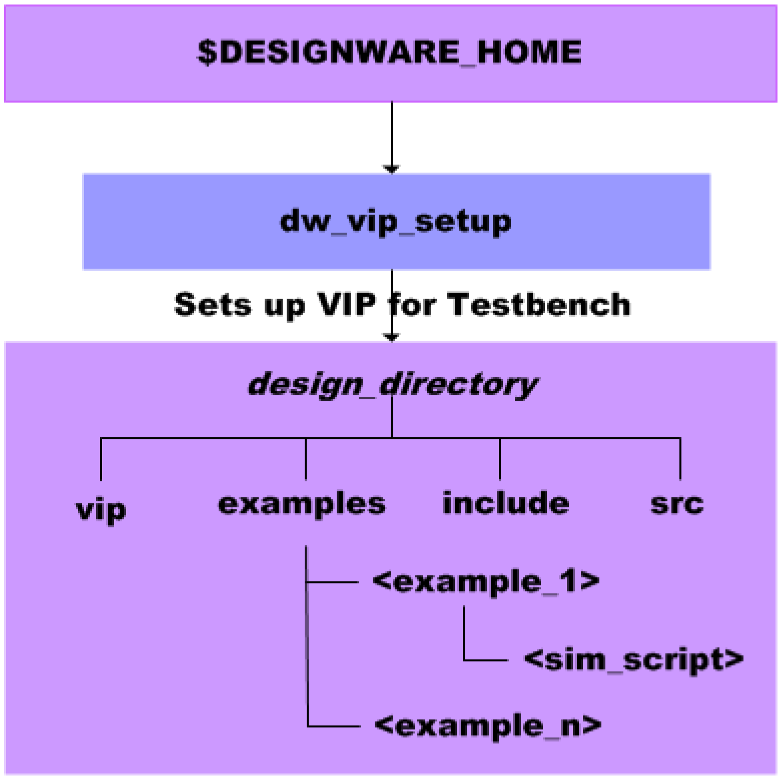
- "Creating a Testbench Design Directory" on page 18
- "The dw_vip_setup Utility" on page 19

2.9.1 Creating a Testbench Design Directory

A design directory contains a version of the VIP that is set up and ready to use in a testbench. The `dw_vip_setup` utility is used to create the design directories. For more information on `dw_vip_setup`, see "The dw_vip_setup Utility" on page 19.

A design directory gives you the control over the version of VIP in your testbench as it is isolated from the `DESIGNWARE_HOME` installation. You can use `dw_vip_setup` to update the VIP in your design directory. Figure 2-2 shows this process and the contents of a design directory.

Figure 2-2 Design Directory Created by dw_vip_setup



A design directory contains the following sub-directories:

examples	Each VIP includes example testbenches. The `dw_vip_setup` utility adds them in this directory, along with a script for simulation. If an example testbench is specified on the command line, this directory contains all the files required for model, suite, and system testbenches.
include	The language-specific include files that contain the critical information for VIP models. This directory is specified in simulator command lines.
src	The VIP-specific include files (not used by all VIP). This directory may be specified in simulator command lines.
.dw_vip.cfg	A database of all the VIP models used in the testbench. The `dw_vip_setup` utility reads this file to rebuild or recreate a design setup.

Note Do not modify this file because `dw_vip_setup` depends on the original content.

2.9.2 The dw_vip_setup Utility

The `dw_vip_setup` utility provides the following features:

- Adds, removes, or updates VIP models in a design directory
- Adds example testbenches to a design directory, the VIP models they use (if necessary), and creates a script for simulating the testbench using any of the supported simulators
- Restores (cleans) example testbench files to their original state
- Reports information about your installation or design directory, including version information
- Supports Protocol Analyzer (PA)
- Supports the FSDB wave format

This section consists of the following sub-sections:

- "Setting Environment Variables" on page 20
- "The dw_vip_setup Command" on page 20

2.9.2.1 Setting Environment Variables

Before running `dw_vip_setup`, the `DESIGNWARE_HOME` environment must point to the location where the VIP is installed.

2.9.2.2 The dw_vip_setup Command

From the command prompt, invoke `dw_vip_setup`. The `dw_vip_setup` command checks command-line argument syntax and makes sure that the requested input files exist. The general form of the command is as follows:

```
% dw_vip_setup [-p[ath] directory] switch (model [-v[ersion] latest | version_no] )
```

or

```
% dw_vip_setup [-p[ath] directory] switch -m[odel_list] filename
```

where,

[**-p**[ath] *directory*]	The optional `-path` argument specifies the path to your design directory. When omitted, `dw_vip_setup` uses the current working directory.
switch	The `switch` argument defines the `dw_vip_setup` operation.

Table 2-1 lists the switches and their applicable sub-switches.

Table 2-1 Setup Program Switch Descriptions

Switch	Description
-a[dd] (model [-v[ersion] version]) …	Adds the specified model or models to the specified design directory or current working directory. If you do not specify a version, the latest version is assumed. The model names are as follows: • uart_agent_svt • uart_txrx_svt • uart_monitor_svt The -add switch makes dw_vip_setup to build suite libraries from the same suite as the specified models, and to copy the other necessary files from $DESIGNWARE_HOME.
-r[emove] model	Removes all versions of the specified model or models from the design. The dw_vip_setup program does not attempt to remove any include files used solely by the specified model or models. The model names are as follows: • uart_agent_svt • uart_txrx_svt • uart_monitor_svt
-u[pdate] (model [-v[ersion] version]) …	Updates to the specified model version for the specified model or models. The dw_vip_setup script updates to the latest models when you do not specify a version. The model names are as follows: • uart_agent_svt • uart_txrx_svt • uart_monitor_svt The -update switch causes dw_vip_setup to build suite libraries from the same suite as the specified models, and to copy the other necessary files from $DESIGNWARE_HOME.
-e[xample] {scenario | model/scenario} [-v[ersion] version]	The dw_vip_setup script configures a testbench example for a single model or a system testbench for a group of models. The program creates a simulator run program for all the supported simulators. If you specify a scenario (or system) example testbench, the models needed for the testbench are included automatically and do not need to be specified in the command. Note: Use the -info switch to list all the available system examples.
-ntb	Not supported.

Table 2-1 Setup Program Switch Descriptions (Continued)

Switch	Description
-svtb	Use this switch to set up models and example testbenches for SystemVerilog. The resulting design directory is streamlined and can only be used in SystemVerilog simulations.
-c[lean] {scenario | model/scenario}	Cleans the specified scenario/testbench in either the design directory (as specified by the -path switch) or the current working directory. This switch deletes all files in the specified directory, then restores all Synopsys-created files to their original contents.
-i/nfo design | home[:<product>[:<version>[:<methodology>]]]	Generate an informational report on a design directory or VIP installation. `design`: If the '`-info design`' switch is specified, the tool displays product and version content within the specified design directory to standard output. This output can be captured and used as a modellist file for input to this tool to create another design directory with the same content. `home`: If the '`-info home`' switch is specified, the tool displays product, version, and example content within the VIP installation to standard output. Optional filter fields can also be specified such as <product>, <version>, and <methodology> delimited by colons (:). An error will be reported if a nonexistent or invalid filter field is specified. Valid methodology names include: OVM, RVM, UVM, VMM, and VLOG.
-h[elp]	Returns a list of valid dw_vip_setup switches and the correct syntax.
model	The UART VIP models are as follows: • uart_agent_svt • uart_txrx_svt • uart_monitor_svt The model argument defines the model or models that dw_vip_setup acts upon. This argument is not needed with the -info or -help switches. All switches that require the model argument may also use a model list. You may specify a version for each listed model, using the -version option. If omitted, dw_vip_setup uses the latest version. The -update switch ignores model version information.
-m[odel_list] filename	The -model_list argument causes dw_vip_setup to use a user-specified file to define the list of models on which the program acts. The model_list, like the model argument, can contain model version information. Each line in the file contains the following syntax: model_name [-v version] –or– # Comments
-b/ridge	Updates the specified design directory to reference the current DESIGNWARE_HOME installation. All product versions contained in the design directory must also exist in the current DESIGNWARE_HOME installation.

Table 2-1 Setup Program Switch Descriptions (Continued)

Switch	Description
-pa	Enables the run scripts and Makefiles generated by dw_vip_setup to support PA. If this switch is enabled, and the testbench example produces XML files, PA will be launched and the XML files will be read at the end of the example execution. For run scripts, specify `-pa`. For Makefiles, specify `-pa = 1`.
-waves	Enables run scripts and Makefiles generated by dw_vip_setup to support the `fsdb` waves option . To support this capability, the testbench example must generate an FSDB file when compiled with the WAVES Verilog macro set to `fsdb`, that is, `+define+WAVES=\"fsdb\"`. If a .fsdb file is generated by the example, the Verdi nWave viewer will be launched. For run scripts, specify `-waves fsdb`. For Makefiles, specify `WAVES=fsdb`.
-doc	Creates a doc directory in the specified design directory which is populated with symbolic links to the `DESIGNWARE_HOME` installation for documents related to the given model or example being added or updated.
-methodology <name>	When specified with -doc, only documents associated with the specified methodology name are added to the design directory. Valid methodology names include: OVM, RVM, UVM, VMM, and VLOG.
-copy	When specified with -doc, documents are copied into the design directory, not linked.
`-simulator <vendor>`	When used with the `-example` switch, only simulator flows associated with the specified vendor are supported with the generated run script and Makefile. **Note** Currently the vendors VCS, MTI, and NCV are supported.

Note The `dw_vip_setup` utility treats all lines beginning with "#" as comments.

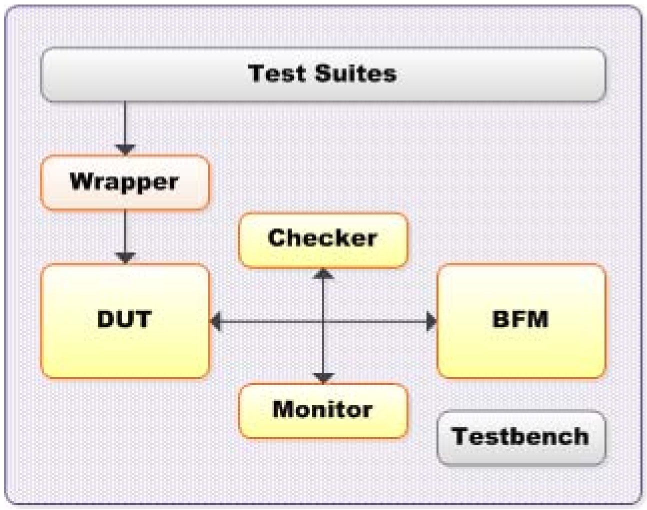
3 General Concepts

This chapter consists of the following sections:

3.1 Architectural Overview

Figure 3-1 shows the internal block diagram of the UART VIP.

Figure 3-1 UART VIP: Architectural Overview



3.2 Integrating With the DUT

This section gives an overview of the following steps that are used to integrate the UART VIP with the DUT:

- "Modifying User Parameters" on page 26
- "Modifying Signal Names" on page 27
- "Instantiating the DUT in the VIP Verification Environment" on page 27

3.2.1 Modifying User Parameters

This section consists of the following sub-section:

- "Port Interface" on page 26

3.2.1.1 Port Interface

The modules of the UART VIP have various ports. All the signals, except `clk`, are active-low signals Refer Tables 3-1 for these ports.

Table 3-1 Port Interface

#	Port	Direction	Description
1.	rst	in	This port is for RESET. It supports the positive edge reset.
2.	clk	in	This port is for the clock that is used as a reference clock must at 10GHz.
3.	sin	DCE to DTE	This port is used as a serial data port. This is an output from DCE and an input to DTE.
4.	sout	DTE to DCE	This port is used as a serial data port. This is an output from DTE and an input to DCE.
5.	dsr	DCE to DTE	This port is used as a data set ready port. This is an output from DCE and an input to DTE.
6.	cts	DCE to DTE	This port is used as a Clear to Send (CTS) port. This is an output from DCE and an input to DTE.
7.	dtr	DTE to DCE	This port is used as a data-terminal ready port. This is an output from DTE and an input to DCE.
8.	rts	DTE to DCE	This port is used as a Request to Send (RTS) port. This is an output from DTE and an input to DCE.

3.2.2 Modifying Signal Names

To modify the names of the signals that are connected to the VIP to match the DUT, you need to modify the `top.v` file appropriately.

3.2.3 Instantiating the DUT in the VIP Verification Environment

The sample code required to instantiate the DUT in the VIP verification environment is available in the `top.v` file.

The sample code should be copied and pasted into the file containing the testbench of the VIP verification environment while making appropriate modifications.

3.3 Handshaking Connections

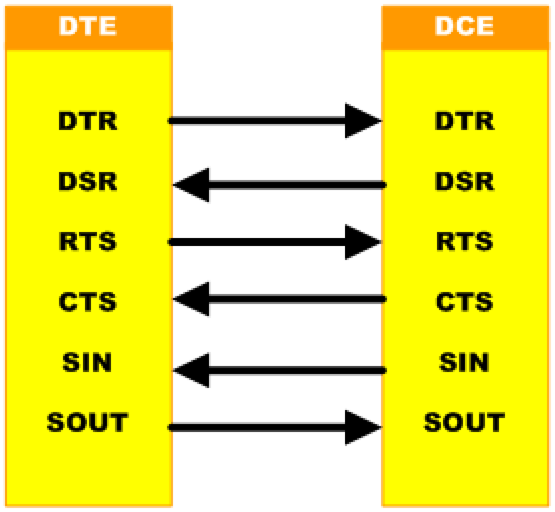
The different handshaking connections possible using the UART VIP are as follows:

- ✦ "Full-Hardware Handshaking" on page 27
- ✦ "Without-Hardware Handshaking" on page 29
- ✦ "Software Handshaking" on page 30

3.3.1 Full-Hardware Handshaking

In this handshaking, both RTS-CTS and DTR-DSR signals are used to exchange flow control information. In addition, data ports are connected to each other. Figures 3-2 shows the standard interface between DTE and DCE.

Figure 3-2 Full Hardware Handshaking



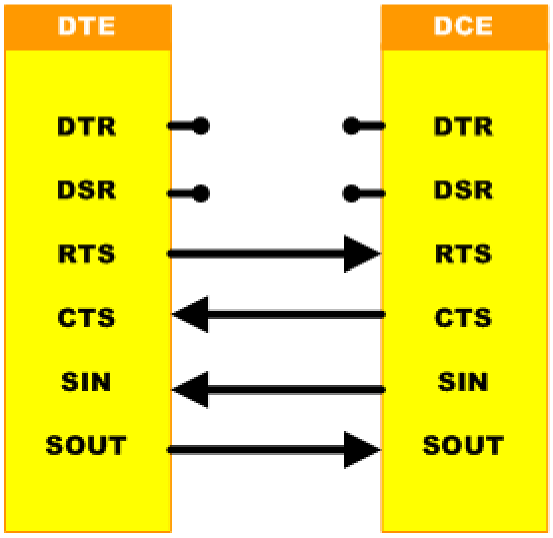
In the partial- hardware handshaking, RTS-CTS signals can be connected to perform the following types of handshaking:

- ✦ "Handshaking in a Single Direction (DTE --> DCE)"
- ✦ "Handshaking in Both Directions"

3.3.1.1 Handshaking in a Single Direction (DTE --> DCE)

In this handshaking, only RTS-CTS signals are used to exchange flow control information. DTR-DSR signals are left unconnected. Refer Figures 3-3 for the same.

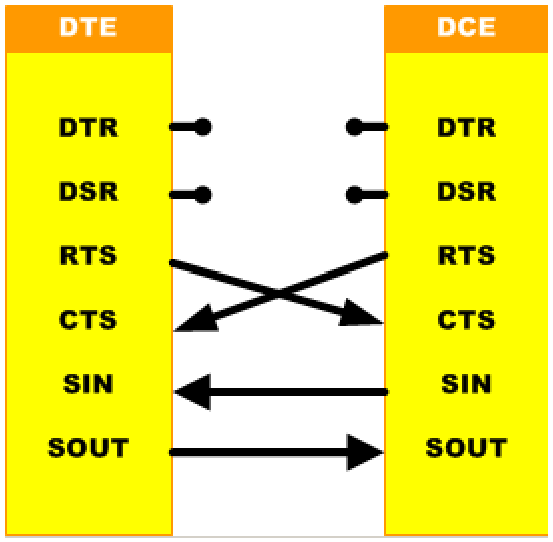
Figure 3-3 Hardware Handshaking in a Single Direction



3.3.1.2 Handshaking in Both Directions

In this handshaking, only RTS-CTS signals are cross-connected to exchange flow control information in both the directions. Figures 3-4 shows DTR-DSR signals that are left unconnected.

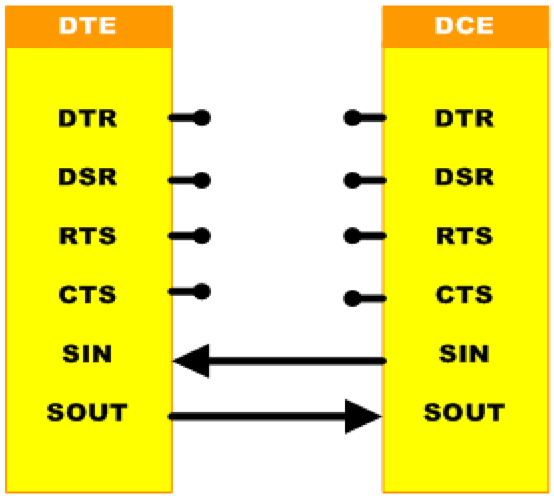
Figure 3-4 Handshaking in Both Directions



3.3.2 Without-Hardware Handshaking

In this handshaking, only data ports are connected to each other. All the other ports have no connection. Figures 3-5 is the connection where RTS-CTS and DTR-DSR handshaking is not required.

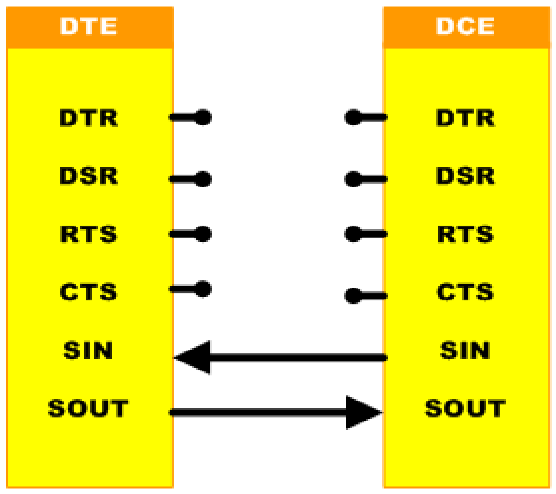
Figure 3-5 Without Hardware Handshaking



3.3.3 Software Handshaking

In this handshaking, only data ports are connected to each other. XON and XOFF data characters are used for flow control. Figures 3-6 shows the software handshaking.

Figure 3-6 Software Handshaking



3.4 BFM

The UART BFM provides the following features:

- Generates and drives bus traffic as a UART.
- Asynchronous RS232 character-based transmit or receive function.
- Full-duplex operation, that is, capable of transmitting and receiving simultaneously.
- On chip baud-rate generator 1 to 65535 divisor generates 16X clock.
- Modem interface
- Configurable Baud Rate Divisor, that is, you can select the baud rate divisor according to the UART DUT requirement.
- Configurable Fractional Baud Rate Divisor, that is, you can select the fractional baud rate divisor according to the UART DUT requirement.
- Simple and flexible BFM tasks:
  - do_cmd() for commands and parameters that change with every command. The do_cmd provides the following command feature:
    - Number of data bytes involved in current transactions.
  - do_cfg() for configuring BFM parameters with the following features:
    - Configures the BFM for the buffer size of the receiver.
    - Configures the BFM for the break condition when the BFM transmits all zero packets.
    - Configures the BFM for the delay time of the Request to Send (RTS) assertion.
    - Configures the BFM for the delay time of the Clear to Send (CTS) assertion.
    - Configures the BFM for an inter-cycle delay time.
- do_cfg_rd() to read BFM parameters.

- do_pkt() to pass data to BFMs.
- do_err() to inject errors with the following features:
  - Occurrence of invalid parity
  - Occurrence of invalid stop bits
  - Occurrence of the break condition

The section covers the following BFM tasks:

- "The do_cmd Task" on page 31
- "The do_cfg Task" on page 32
- "The do_err Task" on page 37
- "The do_pkt Task" on page 38
- "The do_cfg_rd Task" on page 38

3.4.1 The do_cmd Task

This task sets the command of the BFM to a specified value. The task parameters are restricted to only those parameters that change with every command. All other parameters can be changed by using the do_cfg command. The task allows you to initiate a transaction through the BFM on the UART bus.

The task has the following syntax:

```
do_cmd(packet_count);
```

Tables 3-2 shows the arguments for the do_cmd task.

Table 3-2 Arguments for the do_cmd Task: BFM

Argument	Argument Value	Description
packet_count	integer	Specifies the number of packets to transmit. It should not exceed 2000.

For example, to transmit 10 packets of data by BFM0, use the following command:

```
`BFM0.do_cmd(10);
```

3.4.2 The do_cfg Task

This task configures BFM parameters. The do_cfg task specifies the following parameter:

✦ Modes: All parameters have the SVT_UART_MODE prefix.

Note For error injection, use the do_err command.

The do_cfg task has the following syntax:

```
do_cfg(param, pvalue);
```

Tables 3-3 shows the arguments for the do_cfg task.

Table 3-3 Arguments for the do_cfg Task: BFM

Argument	Argument Value	Description
param	integer	A constant that determines the model's operating conditions that are affected by executing this command. Tables 3-4 shows the possible values.
pvalue	integer	An integer that specifies the new value of the operating condition specified by the value parameter. Tables 3-4 shows the possible values.

Tables 3-4 shows the values of pvalue for the do_cfg task.

Table 3-4 Values of pvalue for the do_cfg Task: BFM

Parameter	Data Type	Default value	Description
Modes			
SVT_UART_MODE_SEV_CHG	integer	SVT_UART_ERROR	Sets the severity level of a message that is printed out for an instance.
SVT_UART_MODE_FILE_HANDLE	integer	SVT_UART_FALSE	Specifies the handle of a file that stores the log information.
Functional			
SVT_UART_RECEIVER_BUFFER_SIZE	integer	10	Specifies the receiver-buffer size of DCE.
SVT_UART_BUFFER_FLUSH_DELAY	integer	100	Specifies delay to flush the receiver buffer by DCE.
SVT_UART_INTER_CYCLE_DELAY	integer	200	Specifies the inter-cycle delay.
SVT_UART_DELAY_RTS	integer	200	Specifies delay in the RTS assertion by DTE.

Table 3-4 Values of pvalue for the do_cfg Task: BFM

Parameter	Data Type	Default value	Description
SVT_UART_DELAY_CTS	integer	200	Specifies delay in the CTS assertion by DCE.
SVT_UART_BREAK	1-bit	SVT_UART_BREAK_DIS	Configures the BFM to generate the break condition. SVT_UART_BRAK_EN: Enables the break condition. SVT_UART_BREAK_DIS: Disables the break condition.
SVT_UART_DISABLE_BFM_ERROR	reg	1’b0	Configures the BFM to disable the error reporting. 1’b0: Enables the error reporting. 1’b1: Disables the error reporting.
SVT_UART_DTR_ASSERT_DELAY_TIME	integer	1024	Specifies delay time in ns for the DTR assertion after the RTS assertion.
SVT_UART_SET_PARITY	2-bit reg	3'b001	Specifies the type of parity.
SVT_UART_STOP_BITS	integer	2	Specifies the number of stop bits in a packet.
SVT_UART_DATA_SZ	4-bit reg	9	Specifies the number of data bits in a packet.
SVT_UART_MODE_BAND	1-bit reg	1'b1	Specifies the mode of operation, that is, hardware or software.
SVT_UART_DISABLE_RTS_CTS_HANDSHAKE	1-bit reg	1'b0	Configures the BFM to disable the RTS-CTS handshaking.
SVT_UART_DISABLE_DTR_DSR_HANDSHAKE	1-bit reg	1'b0	Configures the BFM to disable the DTR-DSR handshaking.
SVT_UART_XON_DATA_PATTERN	9-bit reg	1'b0	Sets the XON data pattern in case of the software handshaking.
SVT_UART_XOFF_DATA_PATTERN	9-bit reg	1'b0	Sets the XOFF data pattern in case of the software handshaking.
SVT_UART_SEND_XON_DATA_PATTERN	1-bit reg	11	Sends the XON data pattern with the high priority, that is, it suspends the ongoing data to transmit XON and resumes when it transmits XON.

Table 3-4 Values of pvalue for the do_cfg Task: BFM

Parameter	Data Type	Default value	Description
SVT_UART_SEND_XOFF_DATA_PATTERN	1-bit reg	13	Sends the XOFF data pattern with the high priority, that is, it suspends the ongoing data to transmit XOFF and resumes when it transmits XOFF.
SVT_UART_DTE_WAIT_FOR_XON_AFTR_POWER_UP	1-bit reg	1'b0	Specifies whether DTE waits for a XON packet before starting any data transmission.
SVT_UART_ENABLE_TX_RX_HANDSHAKE	1-bit reg	1'b0	Set this parameter to 1’b1 to enable both TX and RX handshaking. By default, only TX or RX handshaking is enabled.
SVT_UART_DRIVE_DSR_SIGNAL	-	-	Call this configuration to drive the DSR signal with 0 or 1 accordingly.
SVT_UART_DRIVE_DTR_SIGNAL	-	-	Call this configuration to drive the DTR signal with 0 or 1 accordingly.
SVT_UART _IS_ACTIVE	1-bit reg	1'b0	Set the value to 1’b1 or 1’b0 to configure the BFM instance as ACTIVE or PASSIVE respectively.
SVT_UART _BAUD_DIVISOR	integer	1	Specifies the divisor for dividing the input-clock frequency to generate desired baud clock.
SVT_UART_DEVICE_TYPE_DTE	-	DTE	Sets the device type for the BFM. The valid values are SVT_UART_DCE and SVT_UART_DTE.
SVT_UART_ENABLE_FRACTIONAL_BRD	integer	0	Enables the fractional baud rate generator to generate the baudX16 clock. If set to 1: Enables the fractional baud rate generator 0 : Disables the fractional baud rate generator

Table 3-4 Values of pvalue for the do_cfg Task: BFM

Parameter	Data Type	Default value	Description
SVT_UART_FRACTIONAL_DIVISOR	integer	5	This configuration is relevant only when SVT_UART_ENABLE_FRACTIONAL_BRD is set to 1. It Specifies the fractional part of the divisor value to divide input/reference clock for generating baudX16 clock. For example, if the value of SVT_UART_BAUD_DIVISOR is 1 and the value of SVT_UART_FRACTIONAL_DIVISOR is 5, then the divisor value will be 1.5
SVT_UART_FRACTIONAL_DIVISOR_PERIOD	integer	256	This configuration is relevant only when SVT_UART_ENABLE_FRACTIONAL_BRD is set to 1. It Specifies the number of baudX16 clock cycles over which the value of SVT_UART_FRACTIONAL_DIVISOR holds true. The division of input/reference clock is based on the value of 'mult' and 'SVT_UART_BAUD_DIVISOR', where mult is SVT_UART_FRACTIONAL_DIVISOR * SVT_UART_FRACTIONAL_DIVISOR_PERIOD. For each of the 'mult' number of baudX16 cycles, the reference clock is divided by 'SVT_UART_BAUD_DIVISOR+1', and for each of the 'SVT_UART_FRACTIONAL_DIVISOR_PERIOD-mult' number of baudX16 cycles the reference clock is divided by 'SVT_UART_BAUD_DIVISOR'. For example, if SVT_UART_FRACTIONAL_DIVISOR is 5 and SVT_UART_FRACTIONAL_DIVISOR_PERIOD is 16, then mult is 0.5*16, i.e. 8

Table 3-4 Values of pvalue for the do_cfg Task: BFM

Parameter	Data Type	Default value	Description
SVT_UART_FRACTIONAL_MULT_MEDIAN	integer	5	This configuration is relevant only when SVT_UART_ENABLE_FRACTIONAL_BRD is set to 1. It Specifies the fractional value over which the value of 'mult' (where mult is SVT_UART_FRACTIONAL_DIVISOR * SVT_UART_FRACTIONAL_DIVISOR_PERIOD) will be rounded off to the nearest integer. For example, if the value of SVT_UART_FRACTIONAL_MULT_MEDIAN is 5, and the value of mult comes out to be 6.51, then mult will be rounded off to 7, as the value of SVT_UART_FRACTIONAL_MULT_MEDIAN will be considered as 0.50. If the value of mult is less than or equal to 6.50, then mult will be rounded off to 6.
SVT_UART_ALL_DO_CFG_DONE	integer	1	It is called after all the do_cfg are called. It is required to be called (only after passing all other do_cfgs) when FBRD feature is being used.As a task it is called internally in which the value of ‘MULT’ is calculated by the BFM to generate Baud clock in accordance with the FBRD feature.

Note All delays are with reference to the baudX16 clock.

3.4.3 The do_err Task

The task configures the BFM for injecting errors in the VIP traffic. These error-injected traffic perform the negative testing of the DUT and observe its behavior in response to errors. The task generates the following type of errors:

- Occurrence of invalid parity
- Occurrence of invalid stop bits
- Occurrence of the break condition

The task has the following syntax:

```
do_err(err_name,pvalue);
```

Tables 3-5 shows the arguments for the do_err task.

Table 3-5 Arguments for the do_err Task: BFM

Argument	Argument Value	Description
err_name	integer	A constant that determines which error is to injected. Tables 3-6 shows the possible values.
pvalue	integer	Specifies the value that is used to create an error. Tables 3-6 shows the possible values.

Tables 3-6 shows the arguments for the do_err task.

Table 3-6 Values of pvalue for the do_err Task: BFM

#	Rule	pvalue	Description
1.	SVT_UART_INVAL_PARITY	1-bit	Injects an invalid parity by sending the opposite parity.
2.	SVT_UART_INVAL_STOP_BITS	1-bit	Injects invalid stop bits by transmitting logic 1'b0 in place of stop bits.
3.	SVT_UART_BREAK_CONDITION	1-bit	Injects the break condition in a transaction by passing all zero packets.

For example:

```
do_err(`SVT_UART_INVAL_STOP_BITS, 1'b1);
```

3.4.4 The do_pkt Task

The task creates a data packet in the BFM based on the data specified by users. This task passes the data to be sent by the initiator in case of write commands and passes the expected data in case of read commands.

The task has the following syntax:

```
do_pkt(index, data);
```

Tables 3-7 shows the arguments for the do_pkt task.

Table 3-7 Arguments for the do_pkt Task: BFM

Argument	Argument Value	Description
index	integer	Specifies the location of a byte.
data	integer	Specifies the data to be stored in the index.

For example, to pass 5 bytes for a byte, use the following code snippet:

```
do_pkt(0, 8'h00);
do_pkt(1, 8'h00);
do_pkt(2, 8'h00);
do_pkt(3, 8'h00);
do_pkt(4, 8'h00);
```

3.4.5 The do_cfg_rd Task

The task reads the value of a BFM parameter. The task has the following syntax:

```
do_cfg_rd(param, pvalue);
```

Tables 3-8 shows the arguments for the do_cfg_rd task

Table 3-8 Arguments for the do_cfg_rd Task: BFM

Argument	Argument Value	Description
param	integer	A constant that determines the model's operating conditions that are affected by executing this command.
pvalue	integer	A 32-bit vector that specifies the new value of the operating condition specified by the value parameter. The task returns pvalue, which is 32-bit and you should use only the number of bits specified in pvalue and argument values.

Tables 3-9 shows the arguments for the do_cfg_rd task.

Table 3-9 Values of pvalue for the do_cfg_rd Task: BFM

Parameter	Data Type	Default value	Description
SVT_UART_MODE_SEV_CHG	integer	SVT_UART_ERROR	Sets the severity level of a message that is printed out for an instance.
SVT_UART_MODE_FILE_HANDLE	integer	SVT_UART_FALSE	Specifies the handle of a file that stores the log information.
SVT_UART_RECEIVER_BUFFER_SIZE	integer	10	Specifies the receiver-buffer size of DCE.
SVT_UART_BUFFER_FLUSH_DELAY	integer	100	Specifies delay to flush the receiver buffer by DCE.
SVT_UART_INTER_CYCLE_DELAY	integer	200	Specifies the inter-cycle delay.
SVT_UART_DELAY_RTS	integer	200	Specifies delay in the RTS assertion by DTE.
SVT_UART_DELAY_CTS	integer	200	Specifies delay in the CTS assertion by DCE.
SVT_UART_BREAK	1-bit	SVT_UART_BREAK_DIS	Configures the BFM to generate the break condition. SVT_UART_BRAK_EN: Enables the break condition. SVT_UART_BREAK_DIS: Disables the break condition.
SVT_UART_DTR_ASSERT_DELAY_TIME	integer	1024	Specifies delay time in ns for the DTR assertion after the RTS assertion.
SVT_UART_STOP_BITS	integer	2	Specifies the number of stop bits in a packet.
SVT_UART_MODE_BAND	1-bit reg	1'b1	Specifies the mode of operation, that is, hardware or software.

For example, to read BREAK if occurred in a transaction, use the following code snippet:

```
do_cfg_rd(`SVT_UART_BREAK,`SVT_UART_BREAK_EN);
```

3.5 Checker

The checker module provides the interface-level protocol-checking capability. The checker supports a set of rules for checking a protocol violation. The checker displays an error message if a rule is violated. Each rule in the checker can be disabled. All protocol rules are enabled, by default.

The checker performs the real-time reporting of errors on the clock in which the end symbol of a packet is received. Disable a rule to print the message text as "Log" instead of "Error".

You can enable or disable each rule in the module individually using the `do_cfg` task.

The do_cfg task has the following syntax:

```
do_cfg(param, pvalue);
```

Tables 3-10 shows the arguments for the do_cfg task.

Table 3-10 Arguments for the do_cfg Task: Checker

Argument	Argument Value	Description
param	integer	A constant that determines the model’s operating conditions that are affected by executing this command. Tables 3-11 shows the possible values.
pvalue	integer	An integer that specifies the new value of the operating condition specified by the value parameter. Tables 3-11 shows the possible values.

Tables 3-11 shows the values of pvalue for the do_cfg task.

Table 3-11 Values of pvalue for the do_cfg Task: Checker

Parameter	Data Type	Default value	Description
Modes			
SVT_UART_MODE_SEV_CHG	integer	SVT_UART_LOG	Sets the severity level of a message that is printed out for an instance. SVT_UART_WARN: Changes the severity level to warning. SVT_UART_ LOG: Changes the severity level to log. SVT_UART_INFO: Changes the severity level to info. SVT_UART_ DBG: Changes the severity level to debug. SVT_UART_ ERR: Changes the severity level to error.

Table 3-11 Values of pvalue for the do_cfg Task: Checker

Parameter	Data Type	Default value	Description
SVT_UART_ENABLE_ALL_RULE	1-bit	-	Specifies whether to enable all rules.
SVT_UART_DISABLE_ALL_RULE	1-bit	-	Specifies whether to disable all rules.
SVT_UART_MODE_FILE_HANDLE	integer	-	Specifies the handle of a file that stores the log information.
SVT_UART_PRINT_ERR_SUMMARY	1-bit	-	Specifies the handle of a file to print the error summary.
SVT_UART_SET_PARITY	2-bit reg	3'b001	Specifies the type of parity.
SVT_UART_DATA_SZ	integer	9	Specifies the number of data bits in a packet.
SVT_UART_STOP_BITS	integer	2	Specifies the number of stop bits in a packet.
SVT_UART_MODE_BAND	1-bit reg	1'b1	Specifies the mode of operation, that is, hardware or software.
SVT_UART_WAIT_FOR_DSR	integer	1000	Specifies the time to wait for DSR.
SVT_UART_DISABLE_RTS_CTS_HANDSHAKE	1-bit reg	1'b0	Configures the checker to disable the RTS-CTS handshaking.
SVT_UART_DISABLE_DTR_DSR_HANDSHAKE	1-bit reg	1'b0	Configures the checker to disable the DTR-DSR handshaking.
SVT_UART_MAX_DELAY_TO_XON_AFTER_XOFF	integer	1000	Configures the checker to maximum delay for XON after it detects XOFF.
SVT_UART_XON_DATA_PATTERN	9-bit reg	1'b0	Sets the XON data pattern in case of the software handshaking.
SVT_UART_XOFF_DATA_PATTERN	9-bit reg	1'b0	Sets the XOFF data pattern in case of the software handshaking.
SVT_UART_SEND_XON_DATA_PATTERN	1-bit reg	11	Sends the XON data pattern with the high priority, that is, it suspends the ongoing data to transmit XON and resumes when it transmits XON.
SVT_UART_SEND_XOFF_DATA_PATTERN	1-bit reg	13	Sends the XOFF data pattern with the high priority, that is, it suspends the ongoing data to transmit XOFF and resumes when it transmits XOFF.
SVT_UART_DTE_WAIT_FOR_XON_AFTR_POWER_UP	1-bit reg	1'b0	Specifies whether DTE waits for a XON packet before starting any data transmission.

Table 3-11 Values of pvalue for the do_cfg Task: Checker

Parameter	Data Type	Default value	Description
SVT_UART_ENABLE_TX_RX_ HANDSHAKE	1-bit reg	1'b0	Set this parameter to 1'b1 to enable both TX and RX handshaking. By default, only TX or RX handshaking is enabled.
SVT_UART _BAUD_DIVISOR	integer	1	Specifies the divisor for dividing the input-clock frequency to generate desired baud clock.
SVT_UART_ENABLE_FRACTI ONAL_BRD	integer	0	Enables the fractional baud rate generator to generate the baudX16 clock. If set to 1 : Enables the fractional baud rate generator 0 : Disables the fractional baud rate generator
SVT_UART_FRACTIONAL_DIVI SOR	integer	5	This configuration is relevant only when SVT_UART_ENABLE_FRACTIONAL_BRD is set to 1. It specifies the fractional part of the divisor value to divide input/reference clock for generating baudX16 clock. For example, if the value of SVT_UART_BAUD_DIVISOR is 1 and the value of SVT_UART_FRACTIONAL_DIVISOR is 5, then the divisor value will be 1.5
SVT_UART_FRACTIONAL_DIVI SOR_PERIOD	integer	256	This configuration is relevant only when SVT_UART_ENABLE_FRACTIONAL_BRD is set to 1. It Specifies the number of baudX16 clock cycles over which the value of SVT_UART_FRACTIONAL_DIVISOR holds true. The division of input/reference clock is based on the value of 'mult' and 'SVT_UART_BAUD_DIVISOR', where mult is SVT_UART_FRACTIONAL_DIVISOR * SVT_UART_FRACTIONAL_DIVISOR_PERIO D. For each of the 'mult' number of baudX16 cycles, the reference clock is divided by 'SVT_UART_BAUD_DIVISOR+1', and for each of the 'SVT_UART_FRACTIONAL_DIVISOR_PERI OD-mult' number of baudX16 cycles the reference clock is divided by 'SVT_UART_BAUD_DIVISOR'. For example, if SVT_UART_FRACTIONAL_DIVISOR is 5 and SVT_UART_FRACTIONAL_DIVISOR_PERIO D is 16, then mult is 0.5*16, that is 8

Table 3-11 Values of pvalue for the do_cfg Task: Checker

Parameter	Data Type	Default value	Description
SVT_UART_FRACTIONAL_MULT_MEDIAN	integer	5	This configuration is relevant only when SVT_UART_ENABLE_FRACTIONAL_BRD is set to 1. It Specifies the fractional value over which the value of 'mult' (where mult is SVT_UART_FRACTIONAL_DIVISOR * SVT_UART_FRACTIONAL_DIVISOR_PERIOD) will be rounded off to the nearest integer. For example, if the value of SVT_UART_FRACTIONAL_MULT_MEDIAN is 5, and the value of mult comes out to be 6.51, then mult will be rounded off to 7, as the value of SVT_UART_FRACTIONAL_MULT_MEDIAN will be considered as 0.50. If the value of mult is less than or equal to 6.50, then mult will be rounded off to 6.
SVT_UART_ALL_DO_CFG_DONE	integer	1	It is called after all the do_cfg are called. It is required to be called (only after passing all other do_cfgs) when the FBRD feature is being used. As a task it is called internally in which the value of 'MULT' is calculated by the BFM to generate Baud clock in accordance with the FBRD feature.

For example, to set the severity level of the `SVT_UART_CHK_PARITY_MISMATCH_U` rule as error, use the following:

```
do_cfg(`SVT_UART_CHK_PARITY_MISMATCH_U,`SVT_UART_ENABLE_RULE);
```

Also, to set the severity level of the `SVT_UART_CHK_PARITY_MISMATCH_U` rule as log:

```
do_cfg(`SVT_UART_CHK_PARITY_MISMATCH_U,`SVT_UART_DISABLE_RULE);
```

This section consists of the following sub-sections:

- ✦ "Checker Rule" on page 44
- ✦ "Events" on page 46

3.5.1 Checker Rule

Tables 3-12 shows the list of rules in the checker:

Table 3-12 Checker Rule

#	Rule	Description*
1.	SVT_UART_CHK_INVAL_STOP_BITS_U	Upstream : Invalid Stop Bits
2.	SVT_UART_CHK_INVAL_STOP_BITS_D	Downstream : Invalid Stop Bits
3.	SVT_UART_CHK_PARITY_MISMATCH_U	Upstream : Invalid Parity Received
4.	SVT_UART_CHK_PARITY_MISMATCH_D	Downstream : Invalid Parity Received
5.	SVT_UART_CHK_CTS_BEFORE_RTS	CTS should not be asserted before RTS is detected
6.	SVT_UART_CHK_SOUT_BY_DTE_BEFORE_HNDSHK	SOUT should not be asserted by DTE before RTS&CTS handshake completes
7.	SVT_UART_CHK_RTS_BEFORE_DTR	RTS should not be asserted by DTE before DTR is asserted
8.	SVT_UART_CHK_CTS_BEFORE_DSR	CTS should not be asserted by DCE before DSR is asserted
9.	SVT_UART_CHK_SIN_BY_DCE_BEFORE_DSR	SIN should not be asserted by DCE before DSR is asserted
10.	SVT_UART_CHK_SIN_BY_DCE_BEFORE_DTR	SIN should not be asserted by DCE before DTR is detected
11.	SVT_UART_CHK_BRK_DET_U	Upstream : Break Detected
12.	SVT_UART_CHK_BRK_DET_D	Downstream : Break Detected
13.	SVT_UART_CHK_XON_NOT_TRANS_U	Upstream : XON Not Transmitted After Xoff Detected
14.	SVT_UART_CHK_INVAL_TRANS_BET_OFF_ON_D	Downstream : Invalid Transaction Between XOFF and XON
15.	SVT_UART_CHK_RTS_BEFORE_DSR	RTS should not be asserted before DSR is detected
16.	SVT_UART_CHK_DTR_DEASSERT_DURING_HNDSHK	DTR should not be Deasserted by DTE during RTS/CTS handshake
17.	SVT_UART_CHK_DSR_DEASSERT_DURING_HNDSHK	DSR should not be Deasserted by DCE during RTS/CTS handshake
18.	SVT_UART_CHK_RTS_DEASSERT_DURING_HNDSHK	RTS should not be Deasserted by DTE before RTS/CTS handshake could complete

Table 3-12 Checker Rule

#	Rule	Description*
19.	SVT_UART_CHK_CTS_BEFORE_DTR	CTS should not be Asserted by DCE before DTR is detected
20.	SVT_UART_CHK_SOUT_BY_DTE_BEFORE_DSR	SOUT should not be asserted by DTE before DSR is asserted
21.	SVT_UART_CHK_SOUT_BY_DTE_BEFORE_DTR	SOUT should not be asserted by DTE before DTR is detected
22.	SVT_UART_CHK_DTR_DEASSERT_DURING_DATA_D	DTR should not be Deasserted during Downstream DATA transfer
23.	SVT_UART_CHK_DSR_DEASSERT_DURING_DATA_D	DSR should not be Deasserted during Downstream DATA transfer
24.	SVT_UART_CHK_DTR_DEASSERT_DURING_DATA_U	DTR should not be Deasserted during Upstream DATA transfer
25.	SVT_UART_CHK_DSR_DEASSERT_DURING_DATA_U	DSR should not be Deasserted during Upstream DATA transfer
26.	SVT_UART_XOFF_WITH_INVAL_PARITY	Upstream : XOFF detected with Invalid Parity
27.	SVT_UART_XON_WITH_INVAL_PARITY	Upstream : XON detected with Invalid Parity
28.	SVT_UART_SIN_NOT_IDLE_WHN_CTS_HIGH	SIN should not be asserted by DCE when CTS is De-asserted
29.	SVT_UART_SOUT_NOT_IDLE_WHN_CTS_HIGH	SOUT should not be asserted by DTE when CTS is De-asserted
30.	SVT_UART_FALSE_START_U	Upstream : False Start bit Detected
31.	SVT_UART_FALSE_START_D	Downstream : False Start bit Detected

Note * indicates that the Description column shows the messages displayed by the checker during the simulation.

3.5.2 Events

Tables 3-13 shows checker events.

Table 3-13 Events: Checker

#	Event	Description*
1.	event_start_detected_u	Start condition detected in upstream
2.	event_start_detected_d	Start condition detected in downstream
3.	event_stop_detected_u	Stop detected in upstream
4.	event_stop_detected_d	Stop detected in downstream

Note * indicates that the Description column shows the event messages displayed by the checker during the simulation.

3.6 Monitor

The monitor module is a passive display of an activity. It can be instantiated in the top-level testbench. The monitor provides the following features:

- ✦ Provides an ASCII text log of the UART traffic in a chronological order.
- ✦ Displays information in a user-friendly format based on the message level. For more details, refer "Message Level".
- ✦ Displays header and data values as binary-byte strings

You can configure monitor parameters using the do_cfg task. The task has the following syntax:

```
do_cfg(param, pvalue);
```

Tables 3-14 shows the arguments for the do_cfg task.

Table 3-14 Arguments for the do_cfg Task: Monitor

Argument	Argument Value	Description
param	integer	A constant that determines the model's operating conditions that are affected by executing this command. Tables 3-15 shows the possible values.
pvalue	integer	An integer that specifies the new value of the operating condition specified by the value parameter. Tables 3-15 shows the possible values.

Tables 3-15 shows the arguments for the do_cfg task.

Table 3-15 Values of pvalue for the do_cfg Task: Monitor

Parameter	Data Type	Default value	Description
SVT_UART_MODE_SEV_CHG	integer	SVT_UART_LOG	Sets the severity level of a message that is printed out for an instance. SVT_UART_WARN: Changes the severity level to warning. SVT_UART_ LOG: Changes the severity level to log. SVT_UART_INFO: Changes the severity level to info. SVT_UART_ DBG: Changes the severity level to debug. SVT_UART_ ERR: Changes the severity level to error.
SVT_UART_MODE_FILE_HANDLE	integer	-	Specifies the handle of a file that stores the log information.
SVT_UART_PRINT_SUMMARY	1-bit	-	Specifies the print summary of the monitor.
SVT_UART_SET_PARITY	2-bit reg	3'b001	Specifies the type of parity.
SVT_UART_DATA_SZ	integer	9	Specifies the number of data bits in a packet.
SVT_UART_STOP_BITS	integer	2	Specifies the number of stop bits in a packet.
SVT_UART_MODE_BAND	1-bit reg	1'b1	Specifies the mode of operation, that is, hardware or software.
SVT_UART_XON_DATA_PATTERN	9-bit reg	1'b0	Sets the XON data pattern in case of the software handshaking.
SVT_UART_XOFF_DATA_PATTERN	9-bit reg	1'b0	Sets the XOFF data pattern in case of the software handshaking.
SVT_UART_ENABLE_TX_RX_HANDSHAKE	1-bit reg	1'b0	Set this parameter to 1’b1 to enable both TX and RX handshaking. By default, only TX or RX handshaking is enabled.
SVT_UART_BAUD_DIVISOR	integer	1	Specifies the divisor for dividing the input-clock frequency to generate desired baud clock.

Table 3-15 Values of pvalue for the do_cfg Task: Monitor

Parameter	Data Type	Default value	Description
SVT_UART_ENABLE_FRACTIONAL_BRD	integer	0	Enables the fractional baud rate generator to generate the baudX16 clock. If set to 1 : Enables the fractional baud rate generator 0 : Disables the fractional baud rate generator.
SVT_UART_FRACTIONAL_DIVISOR	integer	5	This configuration is relevant only when SVT_UART_ENABLE_FRACTIONAL_BRD is set to 1. It Specifies the fractional part of the divisor value to divide input/reference clock for generating BaudX16 clock. For example, if the value of SVT_UART_BAUD_DIVISOR is 1 and the value of SVT_UART_FRACTIONAL_DIVISOR is 5, then the divisor value will be 1.5.

Table 3-15 Values of pvalue for the do_cfg Task: Monitor

Parameter	Data Type	Default value	Description
SVT_UART_FRACTIONAL_DIVISOR_PERIOD	integer	256	This configuration is relevant only when SVT_UART_ENABLE_FRACTIONAL_BRD is set to 1. It Specifies the number of baudX16 clock cycles over which the value of SVT_UART_FRACTIONAL_DIVISOR holds true. The division of input/reference clock is based on the value of 'mult' and 'SVT_UART_BAUD_DIVISOR', where mult is SVT_UART_FRACTIONAL_DIVISOR * SVT_UART_FRACTIONAL_DIVISOR_PERIOD. For each of the 'mult' number of BaudX16 cycles, the reference clock is divided by 'SVT_UART_BAUD_DIVISOR+1', and for each of the 'SVT_UART_FRACTIONAL_DIVISOR_PERIOD-mult' number of baudX16 cycles the reference clock is divided by 'SVT_UART_BAUD_DIVISOR'. For example, if SVT_UART_FRACTIONAL_DIVISOR is 5 and SVT_UART_FRACTIONAL_DIVISOR_PERIOD is 16, then mult is 0.5*16, that is 8.

Table 3-15 Values of pvalue for the do_cfg Task: Monitor

Parameter	Data Type	Default value	Description
SVT_UART_FRACTIONAL_MULT_MEDIAN	integer	5	This configuration is relevant only when SVT_UART_ENABLE_FRACTIONAL_BRD is set to 1. It Specifies the fractional value over which the value of 'mult' (where mult is SVT_UART_FRACTIONAL_DIVISOR * SVT_UART_FRACTIONAL_DIVISOR_PERIOD) will be rounded off to the nearest integer. For example, if the value of SVT_UART_FRACTIONAL_MULT_MEDIAN is 5, and the value of mult comes out to be 6.51, then mult will be rounded off to 7, as the value of SVT_UART_FRACTIONAL_MULT_MEDIAN will be considered as 0.50. If the value of mult is less than or equal to 6.50, then mult will be rounded off to 6.
SVT_UART_ALL_DO_CFG_DONE	integer	1	It is called after all the do_cfg are called. It is required to be called (only after passing all other do_cfgs) when FBRD feature is being used.As a task it is called internally in which the value of ‘MULT’ is calculated by the BFM to generate Baud clock in accordance with the FBRD feature.

For example, to change the file-mode handle, refer the following code snippet:

```
reg[31:0] hnd1;
hnd1 = 0;
monitor.do_cfg(`SVT_UART_MODE_FILE_HANDLE, hnd1);
```

This section consists of the following sub-sections:

- “Events” on page 51
- “Variables” on page 51
- “Message Level” on page 52
- “Output Field Definition” on page 52

3.6.1 Events

The monitor has two modules namely, `inst1` and `inst2`.

`inst1`: Captures downstream transactions, that is, from DTE -> DCE.

`inst2`: Captures upstream transactions, that is, from DCE -> DTE.

Tables 3-16 shows the events of the monitor.

Table 3-16 Monitor: Events

Event	Description
event_mon_pkt_received_over	Indicates that the monitor receives a packet.
event_mon_pkt_start	Indicates the start condition of a packet.
event_mon_pkt_stop	Indicates the stop condition of a packet.

For example, for downstream receiving of packets, use the following event:

```
`UART_MON.inst1.event_mon_pkt_received_over
```

For upstream receiving of packets, use the following event:

```
`UART_MON.inst2.event_mon_pkt_received_over
```

3.6.2 Variables

The monitor has variables to create the checking logic to verify data on the interface.

Tables 3-17 shows the variables of the monitor.

Table 3-17 Monitor: Variables

Variable	Description
data_pattern_xoff	Indicates the value of the XOFF data pattern.
data_pattern_xon	Indicates the value of the XON data pattern.
mon_received_data	Indicates the payload data that the monitor captures on an interface.
mon_received_packet	Indicates a packet frame that the monitor captures on an interface.

3.6.3 Message Level

You can configure the message levels of the monitor based on table by specifying +argument to a simulator on the command-line or in the run script.

Table 3-18 Message Level: Monitor

Argument	Description
SVT_UART_ERROR	Sets the message level to Error. It displays errors that are generated while running test cases. (default: not defined)
SVT_UART_WARNING	Sets the message level to Warning. It displays errors along with warnings. (default: not defined)
SVT_UART_LOG	Sets the message level to Log. It displays error, warning, and packet-level information. (default: defined)
SVT_UART_INFO	Sets the message level to Info. It displays error, warning, and info-level information. (default: not defined)
SVT_UART_DEBUG	Sets the message level to Debug. It displays error, warning, log, and symbol as well as task-level information. (default: not defined)

3.6.4 Output Field Definition

Tables 3-19 shows the output fields of the monitor. It displays unused or invalid fields "-------".

Table 3-19 Output Field

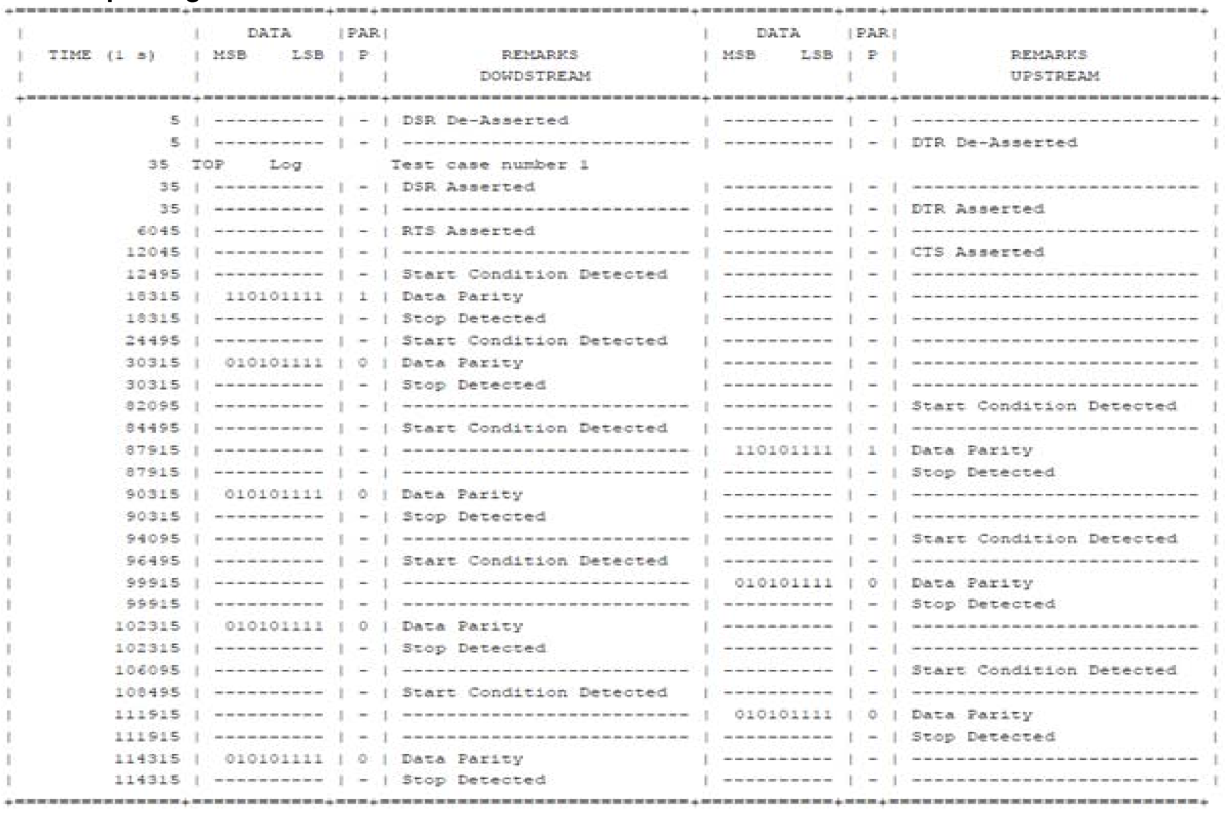
Field	Description
TIME	Specifies the simulation time in which an event occurred.
DATA	Specifies the value of the data byte transmitted in upstream.
PAR	Specifies the parity received in upstream.
REMARKS UPSTREAM	Shows the remarks regarding the data (second field). The possible values are: • RTS asserted • DSR detected • Start-condition detected • Data parity • Stop condition • RTS de-asserted • DSR de-asserted
DATA	Specifies the value of the data byte transmitted in downstream.
PAR	Specifies the parity received in downstream.

Table 3-19 Output Field

Field	Description
REMARKS DOWNSTREAM	Specifies the remarks regarding the data (second field). The possible values are: • RTS asserted • RTS de-asserted • CTS asserted • CTS de-asserted • DSR asserted • DSR de-asserted • DTR asserted • DTR de-asserted • Start-condition detected • Data parity • Stop condition

Figures 3-7 shows the sample log file.

Figure 3-7 Sample Log File



```
+===============+============+===+===========================+============+===+===========================+
|               |    DATA    |PAR|                           |    DATA    |PAR|                           |
|  TIME (1 s)   | MSB    LSB | P |          REMARKS          | MSB    LSB | P |          REMARKS          |
|               |            |   |        DOWDSTREAM         |            |   |         UPSTREAM          |
+===============+============+===+===========================+============+===+===========================+
|             5 | ---------- | - | DSR De-Asserted           | ---------- | - | ------------------------- |
|             5 | ---------- | - | ------------------------- | ---------- | - | DTR De-Asserted           |
             35  TOP    Log        Test case number 1
|            35 | ---------- | - | DSR Asserted              | ---------- | - | ------------------------- |
|            35 | ---------- | - | ------------------------- | ---------- | - | DTR Asserted              |
|          6045 | ---------- | - | RTS Asserted              | ---------- | - | ------------------------- |
|         12045 | ---------- | - | ------------------------- | ---------- | - | CTS Asserted              |
|         12495 | ---------- | - | Start Condition Detected  | ---------- | - | ------------------------- |
|         18315 |  110101111 | 1 | Data Parity               | ---------- | - | ------------------------- |
|         18315 | ---------- | - | Stop Detected             | ---------- | - | ------------------------- |
|         24495 | ---------- | - | Start Condition Detected  | ---------- | - | ------------------------- |
|         30315 |  010101111 | 0 | Data Parity               | ---------- | - | ------------------------- |
|         30315 | ---------- | - | Stop Detected             | ---------- | - | ------------------------- |
|         82095 | ---------- | - | ------------------------- | ---------- | - | Start Condition Detected  |
|         84495 | ---------- | - | Start Condition Detected  | ---------- | - | ------------------------- |
|         87915 | ---------- | - | ------------------------- |  110101111 | 1 | Data Parity               |
|         87915 | ---------- | - | ------------------------- | ---------- | - | Stop Detected             |
|         90315 |  010101111 | 0 | Data Parity               | ---------- | - | ------------------------- |
|         90315 | ---------- | - | Stop Detected             | ---------- | - | ------------------------- |
|         94095 | ---------- | - | ------------------------- | ---------- | - | Start Condition Detected  |
|         96495 | ---------- | - | Start Condition Detected  | ---------- | - | ------------------------- |
|         99915 | ---------- | - | ------------------------- |  010101111 | 0 | Data Parity               |
|         99915 | ---------- | - | ------------------------- | ---------- | - | Stop Detected             |
|        102315 |  010101111 | 0 | Data Parity               | ---------- | - | ------------------------- |
|        102315 | ---------- | - | Stop Detected             | ---------- | - | ------------------------- |
|        106095 | ---------- | - | ------------------------- | ---------- | - | Start Condition Detected  |
|        108495 | ---------- | - | Start Condition Detected  | ---------- | - | ------------------------- |
|        111915 | ---------- | - | ------------------------- |  010101111 | 0 | Data Parity               |
|        111915 | ---------- | - | ------------------------- | ---------- | - | Stop Detected             |
|        114315 |  010101111 | 0 | Data Parity               | ---------- | - | ------------------------- |
|        114315 | ---------- | - | Stop Detected             | ---------- | - | ------------------------- |
+===============+============+===+===========================+============+===+===========================+
```

4 Usage Notes

This chapter consists of the following sections:

4.1 Fractional Baud Rate Divisor

The baud rate defines the transfer rate of the number of symbols per second and it is derived as:

```
Baud rate=(input clock)/(sample rate * divisor)
```

Here, the sample rate of UART is 16 (commonly used value) and the input clock is the peripheral clock.

For example, if the input clock (peripheral clock) = 25 MHz, Baud rate = 9600 and sample rate = 16 then,

```
Divisor value=(25*10^6)/(16*9600)
```

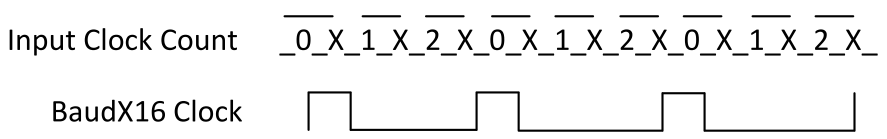
Therefore, `Divisor value = 162.76 (Integer = 162 and Fractional = 0.76)`

4.1.1 Implementation Details

4.1.1.1 Integer Divisor

In case of normal integer values, the baud clock is generated after the Nth cycle of the input clock. For example, the baud clock generated for a divisor of 3, that is N=3 is shown in Figure 4-1:

Figure 4-1 Integer Divisor of 3



4.1.1.2 Fractional Divisor

In case of a fractional baud divisor, the output of baud clock generation will depend on the value of MULT.

There are two ways in which a configurable fractional divisor can be implemented:

4.1.1.2.1 Achieving a Fractional Baud Divisor Value Over Single Bit Period

Take an example where:

```
Reference clock = 25MHz
Uart sampling rate (bit period) = 16
Divisor value = 1.76 (Integer = 1 and Fractional = 0.76) which derives, N (Integer) = 1
MULT = 0.76 * (bit period) = 0.76 * 16 = approx. 12 (exact value is 12.16)
Total duration = bit period = 16
```

Round off the value of MULT based on a configurable real variable, such as the median. If the median is configured to 0.4, then 12.4 or 12.39 will be rounded off to 12 and 12.41 and above will be rounded off to 13.

```
Baud clock=(input clock)/(N + MULT/(bit period))
```

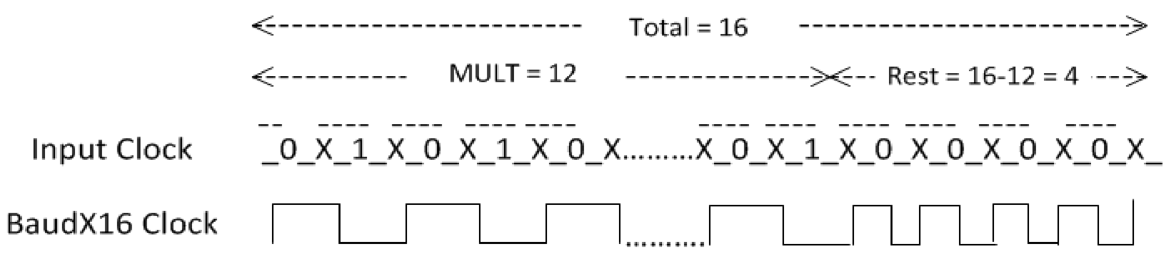
Therefore, `Baud clock = 25 MHz/ (1 + (12/16))`

For the duration of the MULT, each of the output baudX16 clock cycles are generated after 'N+1' input clocks.

For the remaining (Total duration – MULT) output baudX16 clock cycles, each baudX16 clock is generated after 'N' input clocks.

Divisor = 1.76, N=1 and MULT = 12

Figure 4-2 Fractional Baud Divisor Value Over Single Bit Period



As shown in Figure 4-2, 12 baudX16 cycles are generated using (N+1) reference clock cycle, that is two reference clock cycles and the remaining (4 out of 16) baudX16 cycles are generated using N reference clock cycle that is one reference clock cycle, thus giving an approximate average of 1.76 over a bit period.

4.1.1.2.2 Achieving a Fractional Baud Divisor Value Over a Period of (2^REG_WIDTH) Number of BaudX16 Cycles

`REG_WIDTH` is the width of the register that holds the fractional part of the divisor.

Take an example where:

```
Reference clock = 25 MHz
REG_WIDTH = 6
Divisor value = 1.76 (Integer = 1 and Fractional = 0.76)
N = 1;
MULT = 0 .76 * (2^6) = 0.76 * 64 = approx. 48 (exact value is 48.64)
Total duration = bit period = 16
```

Round off the value of MULT based on a configurable real variable, such as the median. If median is configured to 0.4, then 48.4 or 48.39 will be rounded off to 48 and 48.41 and above will be rounded off to 49.

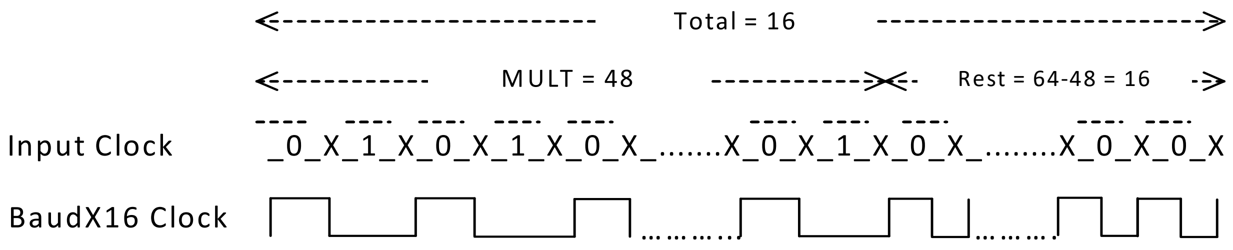
```
Baud clock=(input clock)/(N + MULT/(2^REG_WIDTH))
```

Therefore, `Baud clock = 25 MHz/ (1 + (48/64))`

For the duration of the MULT, each of the output baudX16 clock cycles is generated after 'N+1' input clocks. For the remaining (Total duration – MULT) output baudX16 clock cycles, each baudX16 clock is generated after 'N' input clocks.

Divisor = 1.76 = N=1 and MULT = 48

Figure 4-3 Fractional Baud Divisor Value Over a Period



As shown in Figure 4-3, 48 baudX16 cycles are generated using N+1, that is two reference clock cycles and the remaining (16 out of 64) baudX16 cycles are generated using one reference clock cycle, thus giving an approximate average of 1.76 over 64 baudX16 cycles.

4.2 UART VIP Verilog Integration Reference

4.2.1 Introduction

This section provides the UART VIP top level modification steps which help you integrate the UART VIP into your testbench environment. UART VIP can be used with the following verification topologies:

- ❖ DTE as DUT and DCE as VIP as shown in Figure 4-4
- ❖ DCE as DUT and DTE as VIP as shown in Figure 4-5

Figure 4-4 DTE as DUT and DCE as VIP

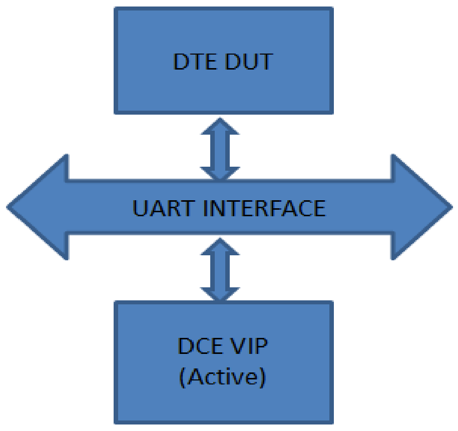
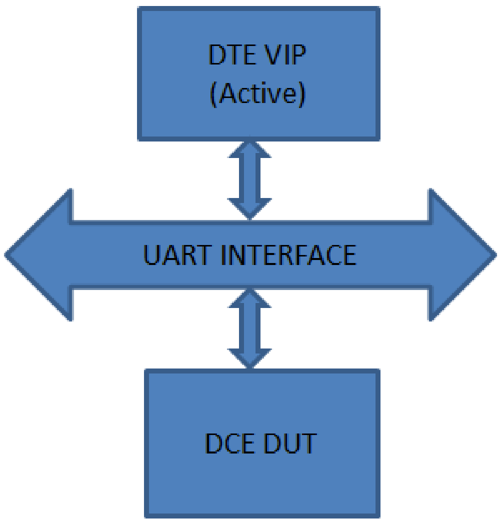


Figure 4-5 DCE as DUT and DTE as VIP



4.2.2 Inclusion of VIP files

To integrate UART VIP into your testbench environment, you need to modify its scripts to include the necessary directories, files and defines by referring the compilation log created after running any one of the tests available as a part of the VIP product.

For example, after you run the `random_test` from the `tb_uart_svt_verilog_basic_sys` example, the `compile.log` file is created in the logs directory. In the `compile.log` file, you can find the commands related to a specific simulator, the details of the directories which need to be included, the defines and other switches.

4.2.3 Integration of UART VIP in a Verilog Based Setup

You can use the following file as a reference to integrate the UART VIP in your testbench environment:

`examples/sverilog/uart_svt/tb_uart_svt_verilog_basic_sys/top.v`

Perform the following steps to integrate the UART VIP into your testbench.

1. Include the `svt_uart_inc.svi` before top level module declaration

   ```
   `include "svt_uart_inc.svi"
   ```

   Note This file includes all the files having BFM and common defines.

2. Declare the system clock, reset and ports relets to DTE or DCE and Monitor and check based on the requirement

   ```
   reg        SystemClock;
   reg        rst;
   /*** Monitor and Checker ports */
   wire       dtr;
   wire       sin;
   wire       cts;
   ```

```
wire      dsr;
wire      rts;
wire      sout;
wire      baudout;

/*** DTE ports */
wire      dte_dtr;
wire      dte_sin;
wire      dte_cts;
wire      dte_dsr;
wire      dte_rts;
wire      dte_sout;

/*** DCE ports */
wire      dce_dtr;
wire      dce_sin;
wire      dce_cts;
wire      dce_dsr;
wire      dce_rts;
wire      dce_sout;
```

3. Define the parameters `SVT_UART_INST_NAME` and `SVT_UART_DEVICE_TYPE` as `DTE/DCE` and `` `SVT_UART_DTE/`SVT_UART_DCE `` respectively using `defparam` based on the requirement and instantiate the `svt_uart_bfm` module as shown in the following code snippet:

```
defparam `test_top.uart_dce.SVT_UART_INST_NAME = "DCE";
defparam `test_top.uart_dce.SVT_UART_DEVICE_TYPE = `SVT_UART_DCE;
svt_uart_bfm   uart_dce(.rst            (rst),
.clk             (SystemClock),
.rts             (dce_rts),
.sout           (dce_sout),
.dsr            (dce_dsr),
.cts            (dce_cts),
.sin            (dce_sin),
.dtr            (dce_dtr),
.baudout   (baudout)
);

/*** Instance of BFM acting as DTE */
defparam `test_top.uart_dte.SVT_UART_INST_NAME = "DTE";
defparam `test_top.uart_dte.SVT_UART_DEVICE_TYPE = `SVT_UART_DTE;
svt_uart_bfm   uart_dte(.rst            (rst),
.clk            (SystemClock),
.rts            (dte_rts),
.sout           (dte_sout),
.dsr            (dte_dsr),
.cts            (dte_cts),
.sin            (dte_sin),
.dtr            (dte_dtr),
.baudout   (baudout)
);
```

Note The instance name can be different.

4. If required, define the parameter `SVT_UART_INST_NAME` as `MON/CHK` for the monitor and checker instance. Instantiate the monitor and checker modules, `svt_uart_mon` and `svt_uart_checker` respectively as shown in the following code snippet:

```
defparam `test_top.uart_mon.SVT_UART_INST_NAME = "MON";
/*** Instance of MONITOR */
svt_uart_mon  uart_mon(.rst             (rst),
.clk             (SystemClock),
.sin             (sin),
.cts             (cts),
.dsr             (dsr),
.dtr             (dtr),
.rts             (rts),
.sout          (sout),
.baudout    (baudout)
);

defparam `test_top.uart_chk.SVT_UART_INST_NAME = "CHK";
/*** Instance of CHECKER */
svt_uart_checker  uart_chk(.rst         (rst),
.clk           (SystemClock),
.sin           (sin),
.cts           (cts),
.dsr           (dsr),
.dtr           (dtr),
.rts           (rts),
.sout        (sout),
.baudout  (baudout)
);
```

5. Connect the DUT's port with the VIP in various verification environments as required. The same goes for the monitor and checker port connections.

   a. RTS Pin of one Device Connected to the CTS Pin of Another Device

Figure 4-6 RTS pin of one device connected to CTS pin of another device

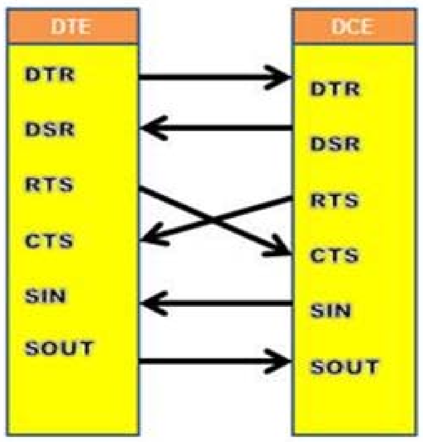


Figure 4-6 shows the cross connection between the RTS pin of one device with the respective CTS pin of another device. To meet this requirement, the VIP must configure the following parameters using the `do_cfg` API:

```
do_cfg (`SVT_UART_MODE_BAND, 0); // in case HARDWARE Handshaking;
```

```
do_cfg (`SVT_UART_ENABLE_TX_RX_HANDSHAKE, 1); // must be set to 1 if cross
connections between RTS and CTS
```

```
do_cfg (`SVT_UART_ENABLE_RTS_CTS_HANDSHAKE, 1); // must be set to 1 if user
needs RTS and CTS to take part in Hardware handshaking otherwise 0
do_cfg (`SVT_UART_ENABLE_DTR_DSR_HANDSHAKE, 1); // must be set to 1 if user
needs DTR and DSR to take part in Hardware handshaking otherwise 0
```

There can be two topologies in this scenario:

i. DTE as DUT and DCE as VIP

In the following code snippet, the DTE is the DUT and the DCE is the VIP and an instance of the DUT is created with the name dut:

```
assign dce_dtr = dut.dtr;
assign dut.dsr = dce_dsr;
assign dce_cts = dut.rts;
assign dut.cts = dce_rts;
assign dce_sout = dut.sout;
assign dut.sin = dce_sin;

assign sin = dce_sin;
assign sout = dut.sout;
assign dtr = dut.dtr;
assign dsr = dce_dsr;
```

ii. DCE as DUT and DTE as VIP

In the following code snippet, the DCE is the DUT and the DTE is the VIP and an instance of the DUT is created with the name dut:

```
assign dut.dtr = dte_dtr;
assign dte_dsr = dut.dsr;
assign dut.cts = dte_rts;
assign dte_cts = dut.rts;
assign dut.sout = dte_sout;
assign dte_sin = dut.sin;

assign sin = dut.sin;
assign sout = dte_sout;
assign dtr = dte_dtr;
assign dsr = dut.dsr;
```

b. RTS and CTS Pin of one Device Connected to Corresponding RTS and CTS Pin of Another Device

Figure 4-7 RTS and CTS Pin of one Device Connected to Corresponding RTS and CTS Pin of Another Device

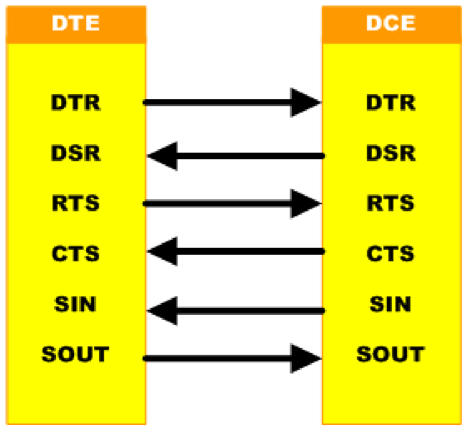


Figure 4-7 shows the RTS pin of the DTE connected to the RTS pin of the DCE and the CTS pin of the DCE connected to the CTS pin of the DTE. To meet this requirement, the VIP must configure the following parameters using the `do_cfg` API:

```
do_cfg (`SVT_UART_MODE_BAND, 0); // in case HARDWARE Handshaking;
do_cfg (`SVT_UART_ENABLE_TX_RX_HANDSHAKE, 0); // must be set to 0
do_cfg (`SVT_UART_ENABLE_RTS_CTS_HANDSHAKE, 1); // must be set to 1 if user
needs RTS and CTS to take part in Hardware handshaking otherwise 0
do_cfg (`SVT_UART_ENABLE_DTR_DSR_HANDSHAKE, 1); // must be set to 1 if user
needs DTR and DSR to take part in Hardware handshaking otherwise 0
```

There can be two topologies in this scenario:

i. DTE as DUT and DCE as VIP

In the following code snippet, the DTE is the DUT and the DCE is the VIP and an instance of the DUT is created with the name dut:

```
assign dce_dtr = dut.dtr;
assign dut.dsr = dce_dsr;
assign dce_rts = dut.rts;
assign dut.cts = dce_cts;
assign dce_sout = dut.sout;
assign dut.sin = dce_sin;
assign sin = dce_sin;
assign sout = dut.sout;
assign dtr = dut.dtr;
assign dsr = dce_dsr;
```

ii. DCE as DUT and DTE as VIP

In the following code snippet, the DCE is the DUT and the DTE is the VIP and an instance of the DUT is created with the name dut:

```
assign dut.dtr = dte_dtr;
assign dte_dsr = dut.dsr;
assign dut.rts = dte_rts;
assign dte_cts = dut.cts;
assign dut.sout = dte_sout;
assign dte_sin = dut.sin;
assign sin = dut.sin;
assign sout = dte_sout;
assign dtr = dte_dtr;
assign dsr = dut.dsr;
```

c. Software Handshaking

Figure 4-8 Software Handshaking

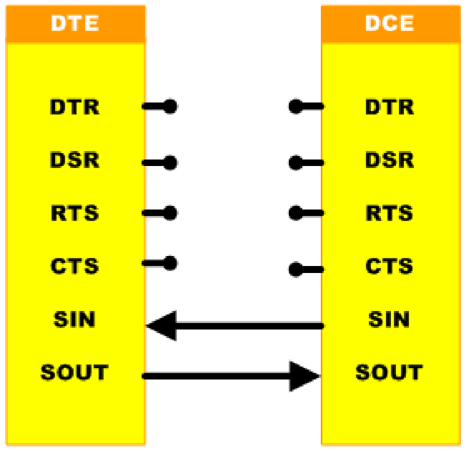


Figure 4-8 shows the connection between the SIN and SOUT pins. Only data pins are used to perform handshaking. To meet this requirement, the VIP must configure the following parameters using the `do_cfg` API:

```
do_cfg (`SVT_UART_MODE_BAND, 1); // in case SOFTWARE Handshaking;
do_cfg (`SVT_UART_ENABLE_TX_RX_HANDSHAKE, 1); // optional to any value
do_cfg (`SVT_UART_ENABLE_RTS_CTS_HANDSHAKE, 0); // must be set to 0 in case of
software handshaking
do_cfg (`SVT_UART_ENABLE_DTR_DSR_HANDSHAKE, 0); // must be set to 0 in case of
software
```

There can be two topologies in this scenario:

i. DTE as DUT and DCE as VIP

In the following code snippet, DTE is the DUT and DCE is the VIP and an instance of the DUT is created with the name dut:

```
assign dce_sout = dut.sout;
assign dut.sin = dce_sin;
assign sin = dce_sin;
assign sout = dut.sout;
```

Note The code snippet takes Figure 4-8 as a reference. The name of the pin could differ for SIN and SOUT. So, while assigning these pins, you need to keep in mind that in case DCE is assigned as the VIP, SIN is the output from the DCE VIP and SOUT is input to the DCE VIP.

ii. DCE as DUT and DTE as VIP

In the following code snippet, DCE is the DUT and DTE is the VIP and an instance of `svt_uart_if` is created with the name `uart_dte_if`. In this case, direct assignment can be made between the DUT's instance and the VIP's interface:

```
assign dut.sout = dte_sout;
assign dte_sin = dut.sin;
assign sin = dut.sin;
assign sout = dte_sout;
```

Note The code snippet takes Figure 4-8 as a reference. The name of the pin could differ for SIN and SOUT. So, while assigning these pins, you need to keep in mind that in case DTE is assigned as the VIP, SOUT is the output from the DTE VIP and SIN is input to the DTE VIP

4.3 UART Verilog Scenario Reference Guide

Table 4-1 provides the scenario reference guide for UART Verilog:

Table 4-1 Verilog Scenario Reference Guide

Scenario	Parameter Setting	Reference Tests	Remarks
To configure different handshaking modes	SVT_UART_MODE_BAND	ts.auto_mode_test.v ts.fbrd_hardwr_hndshk_test.v ts.fbrd_softwr_hndshk_test.v ts.partial_hardwr_hndshk_disable_dtr_dsr_test.v ts.partial_hardwr_hndshk_disable_rts_cts_test.v ts.softwr_hndshk_test.v	Refer do_cfg call in configure_dte, configure_dce and configure_bfm_mon_chk tasks of each test. Possible values are 0 (Hardware Handshake) as default and 1 (Software handshake).
To configure number of stop bits	SVT_UART_STOP_BITS	ts.baud_divisor_test.v ts.data_parity_test.v	Refer do_cfg call in configure_dte, configure_dce and configure_bfm_mon_chk tasks of each test. Possible values are 1 and 2 as default stop bits.
To configure different parity types	SVT_UART_SET_PARITY	ts.data_parity_test.v	Refer do_cfg call in configure_dte, configure_dce and configure_bfm_mon_chk tasks of test. Possible values are: 3'b000-->NO_PARITY 3'b001 --> EVEN_PARITY (default) 3'b010-->ODD_PARITY 3'b011 --> STICK_HIGH_PARITY (MARK) 3'b100 --> STICK_LOW_PARITY (SPACE).

Table 4-1 Verilog Scenario Reference Guide

Scenario	Parameter Setting	Reference Tests	Remarks
To configure different data widths	SVT_UART_DATA_SZ	ts.data_parity_test.v	Refer do_cfg call in configure_dte, configure_dce and configure_bfm_mon_chk tasks of test.Possible values are: 5,6,7,8 and 9(default) bits.
To configure different reciever buffer sizes	SVT_UART_RECEIVER_ BUFFER_SIZ	ts.auto_mode_test.v ts.fbrd_softwr_hndshk_test.v ts.partial_hardwr_hndshk_disable _dtr_dsr_test.v ts.partial_hardwr_hndshk_disable _rts_cts_test.vts.random_test.v ts.softwr_hndshk_test.v	Refer do_cfg call in configure_dte, configure_dce and configure_bfm_mon_chk tasks of each test. Possible values are 1 to 1024 bytes (10 as default).
To configure different inter-cycle delays	SVT_UART_INTER_CYCLE _DELAY	ts.auto_mode_test.v ts.basic_test.v ts.partial_hardwr_hndshk_disable _dtr_dsr_test.v ts.partial_hardwr_hndshk_disable _rts_cts_test.v	Refer do_cfg call in configure_dte, configure_dce and configure_bfm_mon_chk tasks of each test. The default value is 200.
To configure different buffer flush delays	SVT_UART_BUFFER_ FLUSH_DELAY	ts.auto_mode_test.v ts.partial_hardwr_hndshk_disable _dtr_dsr_test.v ts.partial_hardwr_hndshk_disable _rts_cts_test.v	Refer do_cfg call in configure_dte, configure_dce and configure_bfm_mon_chk tasks of each test. The default value is 100.
To Configure XON and XOFF data pattern in case of software handshaking and sending this on high priority	SVT_UART_XON_DATA_ PATTREN	ts.fbrd_softwr_hndshk_test.v ts.softwr_hndshk_test.v	Refer do_cfg call in configure_dte, configure_dce and configure_bfm_mon_chk tasks of each test. The default value is 11.
	SVT_UART_XOFF_DATA_ PATTREN		Refer do_cfg call in configure_dte, configure_dce and configure_bfm_mon_chk tasks of each test. The default value is 13.
	SVT_UART_SEND_XON_ DATA_PATTERN		Refer do_cfg call in configure_dte, configure_dce and configure_bfm_mon_chk tasks of each test.
	SVT_UART_SEND_XOFF_ DATA_PATTERN		Refer do_cfg call in configure_dte, configure_dce and configure_bfm_mon_chk tasks of each test.

Table 4-1 Verilog Scenario Reference Guide

Scenario	Parameter Setting	Reference Tests	Remarks
To insert different error conditions	SVT_UART_INVAL_PARITY SVT_UART_INVAL_STOP_BITS SVT_UART_BREAK_CONDITION	ts.error_test.v ts.error_test.v ts.break_condition_test.v ts.error_test.v	Refer do_err task call in each test.
To model different verification modes	SVT_UART_DISABLE_RTS_CTS_HANDSHAKE SVT_UART_DISABLE_DTR_DSR_HANDSHAKE SVT_UART_ENABLE_TX_RX_HANDSHAKE	ts.auto_mode_test.v ts.partial_hardwr_hndshk_disable_rts_cts_test.v ts.auto_mode_test.v ts.partial_hardwr_hndshk_disable_dtr_dsr_test.v ts.auto_mode_test.v	Refer do_cfg call in configure_dte, configure_dce and configure_bfm_mon_chk tasks of each test.

A
Reporting Problems

A.1 Introduction

This chapter provides you an overview for working through and reporting SVT transactor issues encountered in the field. It describes the data you must submit when a problem is initially reported to Synopsys. After a review of the initial information, Synopsys may decide to request adjustments to the information being requested, which is the focus of the next section.

A.2 Initial Customer Information

Synopsys Support Center. To report a problem, perform the following steps:

1. Before you contact technical support, be prepared to provide the following:
   - A description of the issue under investigation
   - A description of your verification environment
2. Create a Value Change Dump (VCD) file
3. Generate a log file for the simulation

Providing this information ensures the accurate diagnosis of the problem.

If the requested information is not sufficient to determine the cause of your issue, the Synopsys Support Center may request a simple testbench demonstrating the issue. This is the most effective way to debug issues, but if an example cannot be provided, Synopsys may request additional information that could include regenerating the log file using different settings.

If your testbench initializes the random seed (see 'srandom()' in the VCS guide) in the host simulator, then the seed that can be used to demonstrate the problem must be included in the testbench or provided as information for executing the testbench.