

USB 2.0 Host BFM VIP Core

User Guide



System Level Solutions, Inc. (USA)
511 N. Washington Avenue
Marshall, Texas - 75670
(408) 852 - 0067

<https://www.slscorp.com>

IP Core Version: 1.1
Document Version: 1.1
Document Date: August 2021

IP Usage Note

The Intellectual Property (IP) core is intended solely for our clients for physical integration into their own technical products after careful examination by experienced technical personnel for its suitability for the intended purpose.

The IP was not developed for or intended for use in any specific customer application. The firmware/software of the device may have to be adapted to the specific intended modalities of use or even replaced by other firmware/software in order to ensure flawless function in the respective areas of application.

Performance data may depend on the operating environment, the area of application, the configuration, and method of control, as well as on other conditions of use; these may deviate from the technical specifications, the Design Guide specifications, or other product documentation. The actual performance characteristics can be determined only by measurements subsequent to integration.

The reference designs were tested in a reference environment for compliance with the legal requirements applicable to the reference environment.

No representation is made regarding the compliance with legal, regulatory, or other requirements in other environments. No representation can be made and no warranty can be assumed regarding the suitability of the device for a specific purpose as defined by our customers.

SLS reserves the right to make changes to the hardware or firmware or software or to the specifications without prior notice or to replace the IP with a successor model to improve performance or design of the IP. Of course, any changes to the hardware or firmware or software of any IP for which we have entered into an agreement with our customers will be made only if, and only to the extent that, such changes can reasonably be expected to be acceptable to our customers.

Copyright© 2021, System Level Solutions, Inc. (SLS) All rights reserved. SLS, an Embedded systems company, the stylized SLS logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of SLS in India and other countries. All other products or service names are the property of their respective holders. SLS products are protected under numerous U.S. and foreign patents and pending applications, mask working rights, and copyrights. SLS reserves the right to make changes to any products and services at any time without notice. SLS assumes no responsibility or liability arising out of the application or use of any information, products, or service described herein except as expressly agreed to in writing by SLS. SLS customers are advised to obtain the latest version of specifications before relying on any published information and before orders for products or services.

ug_vipusb20hbfm_1.1

Introduction

This guide helps user to know about the basics of USB 2.0 Host BFM VIP Core.

Table below shows the revision history of this user guide.

Version	Date	Description
1.1	August 2021	<ul style="list-style-type: none"> Updated Figure 2-1 and remove Checker section under Bus Activity Inspector in Chp 2 Removed Chp 7 and Appendix D
1.0	August 2014	First Release

How To Find Information

- The Adobe Acrobat Find feature allows you to search the contents of a PDF file. Use Ctrl + F to open the Find dialog box. Use Shift + Ctrl + N to open to the Go To Page dialog box.
- Bookmarks serve as an additional table of contents.
- Thumbnail icons, which provide miniature preview of each page, provide a link to the pages.
- Numerous links shown in Navy Blue color allow you to jump to related information.





How to Contact SLS

For the most up-to-date information about SLS products, go to the SLS worldwide website at <https://www.slscorp.com>. For additional information about SLS products, consult the source shown below.

Information Type	E-mail
Product literature services, SLS literature services, Non-technical customer services, Technical support	support@slscorp.com

Typographic Conventions

The user guide uses the typographic conventions as shown below:

Visual Cue	Meaning
<p>Bold Type with Initial Capital letters</p>	<p>All headings and Sub headings Titles in a document are displayed in bold type with initial capital letters; Example:</p>
<p>Bold Type with Italic Letters</p>	<p>All Definitions, Figure and Table Headings are displayed in Italics. Examples:</p>
<p>1., 2.</p>	<p>Numbered steps are used in a list of items, when the sequence of items is important. such as steps listed in procedure.</p>
<p>•</p>	<p>Bullets are used in a list of items when the sequence of items is not important.</p>
	<p>The hand points to special information that requires special attention</p>
	<p>This caution indicates required information that needs special consideration and understanding and should be read prior to starting or continuing with the procedure or process.</p>
	<p>The warning indicates information that should be read prior to starting or continuing the procedure or processes.</p>
	<p>The feet direct you to more information on a particular topic.</p>

<i>About this Guide</i>	<i>iii</i>
Introduction	iii
How To Find Information	iii
How to Contact SLS	iii
Typographic Conventions	iv
1. Introduction	1
Features	1
2. Architectural Overview	2
Test Suite	2
Transaction Generator	3
Error Injector	3
Host BFM	3
Driver	3
Protocol Layer	3
PHY Signal Generator	3
Host PHY Model	3
Device PHY Model	3
Device DUT	4
Bus Activity Inspector	4
Monitor	4
3. Getting started	5
System Requirements	5
4. Tasks Description	6
cfg_token	7

cfg_data_pkt.....	8
cfg_setup_data	9
cfg_timer	10
cfg_host.....	13
do_cmd.....	14
do_pkt.....	16
inject_err	16
host_status_rd.....	19
5. Events	21
6. USB 2.0 Monitor	23
Log File Header Field Description.....	23
Log File Summary	25
<i>Appendix A: User Defined Tasks</i>	<i>27</i>
initial_seq_task	28
normal_enumeration_tasks	28
fs_enumeration_task	28
Control_Ep_Tests	28
Bulk_In_Tests	28
Bulk_Out_Tests	29
Interrupt_In_Tests	30
Control_Ep_Split_Tests	30
Bulk_In_Split_Tests	30
Bulk_Out_Split_Tests	31
Interrupt_In_Split_Tests	31
Port_Config	32
Defined Files	32
Hub_Enumeration	32
<i>Appendix B: Examples</i>	<i>34</i>
Example 1:	34
Example 2:	35
Example 3:	36
Example 4:	36

Appendix C: Sample Monitor Log File..... 38

The Verification Intellectual Property (Verification IP) is verification model and overall environment, which aids designers and verification engineers in the task of validating the functionality of their design. The Verification IP (VIP) is used in all levels of simulation-based verification. The Verification Intellectual Properties are based on standard protocols used in Networking, computer and system designs such as PCI/PCIx, USB and Ethernet. These components are pre-verified to the standard protocols and contain the necessary infrastructure for test bench generation and checking mechanisms as well as all the appropriate routines to create individual protocols, commonly known as Bus Functional Models (BFM). Test cases greatly aid users in finding functional bugs early in design cycle, hence reducing the overall verification time.

Features

Following are the USB 2.0 Host BFM VIP core features:

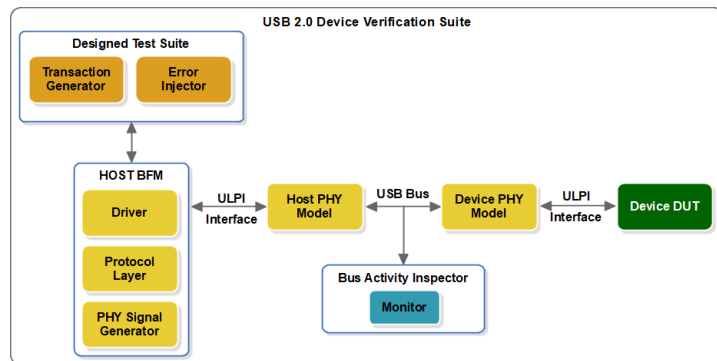
- Compliant to USB2.0 specifications
- Generates and drives bus traffic as a USB Host
- Supports UTMI+(level 3) and D+/D- PHY Interface
- Supports CONTROL and BULK transfer types
- Error insertion
- Control over the timers as per USB 2.0 Specification
- Supports packet level commands
- Extensive event generation
- Simple and flexible BFM tasks
- Supports High Speed and Full Speed device
- Supports INTERRUPT transfer type and SPLIT transfer

2. Architectural Overview

USB2.0 Host BFM model is the heart of USB2.0 Device verification environment. Host model serves the functionality of the host controller IP core which communicates with device IP core which user wants to verify. Model fires various commands/responses to device, inject errors to check device response, decode device responses to verify whether they are proper according to specification or not.

Figure 2-1. illustrates overall architecture of the USB 2.0 Host BFM VIP core.

Figure 2-1. USB 2.0 Host BFM VIP Core Architecture



Each of the blocks is described in detail below:

Test Suite

This is the top block of verification suit which contains various test cases for the bulk in, bulk out and control transactions. Errors are injected, device enumeration and initialization commands are also performed in this block. User can inject required data through various tasks in this block.

Transaction Generator

This block is responsible for generating Bulk, Control, Device initialization and Device enumeration transactions.

Error Injector

In this block, user can inject specific errors to verify the device reaction. User can set more than one errors for single operation.

Host BFM

Host BFM operates at 60 MHz. This block is responsible to execute command operations fired from the “Test Suit” block and send necessary data to phy model of the host. It comprises of driver, protocol layer and phy signals generator blocks.

Driver

This block contains FIFO which stores command fired from the top file then decodes that command and calls protocol layer to perform required operations accordingly.

Protocol Layer

This block contains micro level tasks which are responsible to assemble and disassemble packets and send/receive data over phy signal generator block.

PHY Signal Generator

This block takes data from the protocol layer and generates phy signals required for the phy model to send data over USB Bus.

Host PHY Model

This model takes/sends data from/to phy signal generator block. This model of host is responsible to send data over USB Bus based on ULPI interface protocol. It provides modelling of ULPI/UTMI interface.

Device PHY Model

This model serves the same functionality as host phy model gives. It has ULPI/UTMI interface with the Device IP Core. (DUT)

Device DUT

This is the device IP core which needs to be verified. Based on USB 2.0 specification, it gives response to Host BFM.

Bus Activity Inspector

This block is an independent block of entire verification suit. Its task is to see all the activities happening on USB Bus and if any of activities seems wrong according to USB2.0 Specification, it notifies result with specific errors. It contains monitor and checker blocks.

Monitor

This block is responsible to monitor all bus activities and gives data present on lines, whether they are from DUT side or host BFM side. For more information, refer [Chapter 6. USB 2.0 Monitor](#).

System Requirements

The USB BFM design requires the following hardware and software:

- Hardware Requirements: A PC running the Windows 8/10
- Software Requirements: ModelSim 10.0c or higher

This section describes list of tasks used in the USB 2.0 Device Verification environment. [Table 4-1](#).shows the tasks details.

Table 4-1. Tasks of USB2.0 Device Verification Environment

No.	Task Name	Description
Host BFM Top Module Tasks		
1.	cfg_token	This task configures token packet.
2.	cfg_data_pkt	This task configures data packet parameters.
3.	cfg_setup_data	This task configures setup data bytes for SETUP transaction.
4.	cfg_timer	This task configures different time value used by BFM.
5.	cfg_host	This task configures host BFM for various signalling conditions.
6.	do_cmd	This task generates desired transactions on USB bus.
7.	do_pkt	This task creates user defined data for write requests and expected data for read requests.
8.	inject_err	This task instructs BFM for generating various errors.
9.	host_status_rd	This task is used to read different values of the host BFM model to design few conditional based test cases at the top level.

cfg_token

This task configures token packet. Different parameters and values are given in this task. Host BFM will generate token packet with the use of parameters set by this task. Use define variable present inside the define file for configuration of parameter using this token configuration task. Split operation related configurations are required only when user needs to perform split operation on the HUB DUT. [Table 4-2](#). shows task argument details.

Syntax:

```
cfg_token (Parameter, Parameter Value);
```

Table 4-2. cfg_token

No.	Parameter	Description	Size (Bits)	Parameter Value
1.	TOKEN_PID	Specifies the token PID to be used for token phase of the transaction.	4	USB_OUT_PID: out token USB_IN_PID: in token USB_SOF_PID: sof token USB_SETUP_PID: setup token USB_PING_PID: ping token
2.	DEV_ADDRESS	Specifies device address to perform transaction.	7	Device Address Value should be same as used inside the set address standard request for the device address setting.
3.	EP_NUMBER	Specifies endpoint number of the device.	4	Endpoint value of the device on which transaction will be performed.
4.	FRAME_NUMBER	Specifies frame number.	11	Frame number value for SOF token.
Split Token Configuration Parameters				
5.	SPLIT_HUB_ADDRESS	Specifies the device hub address of the split command.	7	Device hub address.
6.	SPLIT_START_COMPLETE	Specifies start/complete bit during split transaction.	1	USB_TRUE: Start split USB_FALSE: Complete split
7.	SPLIT_HUB_PORT_NUMBER	Specifies port number of the hub for which full/low speed transaction is specified.	7	Port Number

8.	SPLIT_SPEED	Specifies the SPEED for SPLIT transaction.	1	USB_FS: Full speed USB_LS: Low speed
9.	SPLIT_END	Specifies the end field of the split packet.	1	USB_TRUE: end field configured USB_FALSE: end field not configured
10.	SPLIT_EP_TYPE	Specifies endpoint type used in split transaction.	2	USB_CONTROL: Control EP USB_BULK: Bulk EP USB_INTERRUPT: Interrupt EP USB_ISOCH: Isochronous EP

Examples:

1. For setup token PID

```
cfg_token (`TOKEN_PID, USB_SETUP_PID);
```

2. For split full speed

```
cfg_token (`SPLIT_SPEED, USB_FS);
```

cfg_data_pkt

This task configures data packet parameters. Data phase Configuration values will be used by the host BFM to generate a data packet during OUT operation on the device. According to the configuration value data phase of the out operation will be generated. Use do_pkt task to configure data bytes.

[Table 4-3.](#) shows task argument details.

Syntax:

```
cfg_data_pkt (Parameter, Parameter Value);
```

Table 4-3. cfg_data_pkt

No.	Parameter	Description	Size (Bits)	Parameter Value
1.	TRANSFER_TYPE	Specifies the transfer type.	2	USB_CONTROL: Control USB_ISOCH: Isochronous USB_BULK: Bulk USB_INTERRUPT: Interrupt

2.	DATA_PID	Specifies the Data PID to be used.	4	USB_PID_DATA0: Data0 packet USB_PID_DATA1: Data1 packet USB_PID_DATA2: Data2 packet USB_PID_MDATA: MData packet
3.	PCKT_SIZE	Specifies the number of bytes to be transferred during this packet. <i>Note:</i> This should not be more than the maximum packet size of intended endpoint.	16	Packet Size in no of bytes.
4.	EP_MAX_SIZE	Specifies the maximum packet size of intended endpoint.	16	Endpoint max size in bytes. (For future use-optional)

Example:

If TRANSFER_TYPE = USB_CONTROL and DATA_PID = USB_PID_DATA1 then,

```
cfg_data_pkt (` TRANSFER_TYPE, ` USB_CONTROL);
cfg_data_pkt (` DATA_PID, ` USB_PID_DATA1);
```



Current Host BFM supports only Control and Bulk transfers.

cfg_setup_data

This task configures setup data bytes for SETUP transaction. Setup operation is performed on the device with fixed 8 byte data packet inside the data phase. These 8 byte values generate the setup request to be performed on the device. Use defines present inside the define file for parameters as well as to specify its value (where ever possible). [Table 4-4.](#) shows task argument details.

Syntax:

```
cfg_setup_data (Parameters, Parameter Value);
```

Table 4-4. cfg_setup_data

No.	Parameter	Description	Size (Bits)	Parameter Value
1.	SETUP_REQ_DIR	Specifies the direction of data transfer.	1	0: USB_HOST_TO_DEVICE 1: USB_DEVICE_TO_HOST

2.	SETUP_REQ_TYPE	Specifies the request type of the setup command.	2	2'b00: USB_TYPE_STANDARD 2'b01: USB_TYPE_CLASS 2'b10: USB_TYPE_VENDOR 2'b11: USB_TYPE_RESERVED
3.	SETUP_REQ_RECIPIENT	Specifies the setup request recipient.	5	2'b00: USB_REC_DEV 2'b01: USB_REC_INTERFACE 2'b10: USB_REC_ENDPOINT 2'b11: USB_REC_OTHER
4.	SETUP_bREQUEST	Specifies the bRequest.	8	Setup Request
5.	SETUP_wVALUE	Specifies the wValue field of the setup data	16	wValue of Setup Request
6.	SETUP_wINDEX	Specifies the wIndex field of the setup data	16	wIndex of Setup Request
7.	SETUP_wLENGTH	Specifies the wLength field of the setup data	16	Length of data in bytes in data phase of data stage

Example:

For setup request of GET_DEVICE_DESCRIPTOR with 8-bytes of wlength

```

cfg_setup_data (`SETUP_REQ_DIR, `USB_HOST_TO_DEVICE);
cfg_setup_data (`SETUP_REQ_TYPE, `USB_TYPE_STANDARD);
cfg_setup_data (`SETUP_REQ_RECIPIENT,
`USB_REC_DEVICE);
cfg_setup_data (`SETUP_bREQUEST, `GET_DESCRIPTOR);
cfg_setup_data (`SETUP_wVALUE,
{`DESC_TYPE_DEVICE, 8'h0});
cfg_setup_data (`SETUP_wINDEX, 16'h0);
cfg_setup_data (`SETUP_wLENGTH, 16'h8);

```

cfg_timer

This task is used to configure different timers present inside the host BFM model. Configurations gives flexibility of the timing setting to test few error test cases on the device as well as to short the simulation timing of the simulator at the device development stage. List of the different timers are given below. Default values of the timers as set according to the specifications. [Table 4-5.](#) shows task argument details.

Syntax:

`cfg_timer (Parameter, Parameter Value);`

Table 4-5. cfg_timer				
No.	Parameter	Description	Size (Bits)	Parameter Value
Inter Packet Delays				
1.	FS_TX_TX_DELAY	Specifies the delay between two packets sent by Host for full speed transaction.	32	Value in no of clocks
2.	FS_RX_TX_DELAY	Specifies the delay for which host waits before sending packet when it has received packet from device for full speed transaction.	32	Value in no of clocks
3.	FS_TX_RX_DELAY	Specifies the time for which host waits for device response packet to start before timeout for full speed transaction.	32	Value in no of clocks
4.	HS_TX_TX_DELAY	Specifies the delay between two packets sent by Host for high speed transaction.	32	Value in no of clocks
5.	HS_RX_TX_DELAY	Specifies the delay for which host waits before sending packet when it has received packet from device for high speed transaction.	32	Value in no of clocks
6.	HS_TX_RX_DELAY	Specifies the time for which host waits for device response packet to start before timeout for high speed transaction.	32	Value in no of clocks
Reset Signalling				
7.	RESET_TIMER	Duration to drive Reset on bus.	32	Duration Value
8.	CHIRP_VALIDATION_TIMER	Time for which a chirp J or K must be continuously detected by host to consider it valid.	32	Duration Value
9.	RESP_TO_CHIRPK_TIMER	Specifies the time after which host should begin response to K chirp once device has stopped driving Chirp K.	32	Duration Value

10.	DRIVE_CHIRPKJ_WIDTH	Time for which each individual Chirp J or K is to be driven.	32	Duration Value
11.	STOP_CHIRP_DURATION	Time before the end of reset by which host must stop chirp sequence.	32	Duration Value
Resume Timers				
12.	RESUME_K_VALIDATION	Specifies number of clocks for which host must get valid K chirp from the device continuously to consider it as a valid resume request.	32	Duration Value
13.	RESUME_TIMER	Specifies the time for which host drives resume.	32	Duration Value
Suspend Timers				
14.	SUSPEND_TIMER	Specifies the time after which IDLE bus is suspended if suspend feature is enabled.	32	Duration Value
Connection/Disconnection Timers				
15.	CONNECT_VALIDATION_TIME	Time to detect connect event. In other words, this indicates minimum time for which Host must get steady J state to validate device connection.	32	Duration Value
16.	DISCONNECT_VALIDATION_TIME	Time to detect dis-connect event. In other words, this indicates minimum time for which Host must get steady SE0 state to validate device disconnection.	32	Duration Value
Frame Timer				
17.	FRAME_TIMER	Specifies the frame	32	Duration Value
18.	MICRO_FRAME_TIMER	Time to detect the new micro frame interval	32	Duration Value

Example: Configure TX_TX_DELAY time = 11 clocks,
`cfg_timer (`TX_TX_DELAY, 32'd11);`

cfg_host This task configures host BFM for various signalling conditions. [Table 4-6.](#) shows task argument details.

Syntax: `cfg_host (Parameter, Parameter Value);`

Table 4-6. cfg_host				
No.	Parameter	Description	Size (Bits)	Parameter Value
Full/High Speed Switch				
1.	HS_ENABLE	Specifies whether host is to be enabled for high speed operation.	1	0: Full Speed 1: High Speed
Signalling				
2.	AUTO_RESET	Enables host to drive reset signaling for specified number of clocks when device is attached.	1	0: Disable 1: Enable
3.	AUTO_RESUME	Specifies whether host is enabled to drive resume by its own when device initiated resume is detected.	1	0: Disable 1: Enable
4.	AUTO_SUSPEND	Specifies the host to perform the operations of suspend automatically without do_cmd for SUSPEND if bus is IDLE for SUSPEND_TIMER duration.	1	0: Disable 1: Enable
SOF				
5.	AUTO_SOF	Specifies the host to drive SOF automatically without do_cmd for SOF with respect to the duration of FRAME_TIMER.	1	0: Disable 1: Enable
6.	AUTO_FRAME_NUMBER	Specifies auto generation of frame number.	1	0: Disable 1: Enable

Other Parameters				
7.	DATA_TRANSFER_MODE	Specifies the mode of filling the data in the frame.	1	INC: Incremental data (value = 0) RAND: Random data (value = 1)
8.	DATA_CHECKING	Specifies whether IN data is to be checked or not.	1	0: Disable 1: Enable
9.	TRANS_RETRY	Specifies number of transaction retry attempts by host. Host does not retry current transaction if this value is set to zero.	32	Retry Attempts
10.	NAK_RETRY	When NAK_RETRY_COUNTER is enabled, NAK_RETRY specifies number of NAK transaction retry attempts by Host. (Host won't retry if this value is set to zero and NAK_RETRY_COUNTER is enabled.)	4	NAK Retry Attempts
11.	NAK_RETRY_COUNTER	If this counter is enabled then Host will retry the current transaction (In response to NAK from device), "NAK_RETRY" times.	1	0: Disable 1: Enable

Examples:

1. If No. of retry transactions = 3 then,

```
cfg_host (`TRANS_RETRY, 3);
```

2. If NAK retry attempts = 3 then,

```
cfg_host (`NAK_RETRY_COUNTER, `ENABLE);
```

```
cfg_host (`NAK_RETRY, 4'h3);
```

do_cmd

This task generates desired transactions on USB bus. [Table 4-7.](#) shows task argument details.

Syntax:

```
do_cmd (Parameter);
```

<i>Table 4-7. do_cmd</i>		
No.	Parameter	Description
BFM Model Reset		
1.	HOST_BFM_RESET	Initialize all variables to their default values.

Vbus managing Command		
2.	VBUS_ENABLE	This is used to enable VBUS supply.
3.	VBUS_DISABLE	This is used to disable VBUS supply.
Signaling Commands		
4.	USB_RESET	This is used to initiate reset signalling for specified duration.
5.	USB_RESUME	This is used to initiate resume signalling for specified duration.
6.	USB_SUSPEND	This is used to initiate suspend signalling.
Transaction Level Commands		
7.	USB_IN	This is used to perform IN transaction.
8.	USB_OUT	This is used to perform OUT transaction.
9.	USB_SETUP	This is used to perform SETUP transaction.
10.	USB_PING	This is used to perform PING transaction.
SOF/Split Commands		
11.	USB_SOF	This is used to send SOF token.
12.	USB_SPLIT_IN	This is used to perform IN operation with SPLIT token.
13.	USB_SPLIT_OUT	This is used to perform OUT operation with SPLIT token.
14.	USB_SPLIT_SETUP	This is used to perform SETUP operation with SPLIT token.
15.	USB_SPLIT_INT_IN	This is used to perform INTERRUPT IN Transaction with SPLIT Token.

Examples:

1. For setup stage

```
do_cmd (`USB_SETUP);
```

2. For Data IN stage

```
do_cmd (`USB_IN);
```



do_cmd is a non-blocking task. It should be called only after user has entered all required data for desired operation. For example,

1. For OUT Operation:

```
cfg_token (`TOKEN_PID, `USB_OUT_PID);
cfg_token (`DEV_ADDRESS, `DEFAULT_ADDR);
cfg_token (`EP_NUMBER, `DEFAULT_EP);
cfg_data_pkt (`TRANSFER_TYPE, `USB_CONTROL);
cfg_data_pkt (`DATA_PID, `USB_DATA0_PID);
cfg_data_pkt (`PKT_SIZE, `ZERO_LENGTH);
```

```
do_cmd (`USB_OUT);
```

2. For Split IN Operation

```
cfg_token (`TOKEN_PID, `USB_IN_PID);
cfg_token (`DEV_ADDRESS, `DEFAULT_ADDR);
cfg_token (`EP_NUMBER, `DEFAULT_EP);
cfg_data_pkt (`TRANSFER_TYPE, `USB_CONTROL);
cfg_data_pkt (`DATA_PID, `USB_DATA1_PID);
cfg_data_pkt (`PKT_SIZE, 16'h8);
do_cmd (`USB_SPLIT_IN);
```

do_pkt

This task creates user defined data for write requests (OUT transfer) and expected data for read requests (IN transfer). Expected data for read requests will be used by host BFM for data comparison if data checking is enabled.

[Table 4-8.](#) shows task argument details.

Syntax:

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

Table 4-8. do_pkt

No.	Parameter	Description
1.	byte index	Specifies the byte location.
2.	byte value	Specifies the data to be stored in that index.

Example:

For OUT Operation Of 4-bytes and Data transfer mode = Random:

```
do_pkt (0, 8'h01);
do_pkt (1, 8'h02);
do_pkt (2, 8'h04);
do_pkt (3, 8'h06);
```



do_pkt task is used only when data transfer mode is set to **Random**. When data transfer mode is set to **Incremental** then this task is not used. Host BFM will automatically manage and send data incremental data to the device according to the packet size given.

inject_err

This task instructs BFM for generating various errors. Error injection is used to verify the functionality of the device under different error conditions as per the specifications. [Table 4-9.](#) shows task argument details.

Syntax:

```
inject_err (Error Parameter, Error Value);
```

Table 4-9. inject_err				
No.	Parameter	Description	Size (Bits)	Parameter Value
Token Phase				
1.	TKN_INVLD_PID	Set Invalid PID in Token Phase.	4	Wrong Token PID
2.	TKN_PID_CHKSUM_ERR	Set PID checksum error in Token PID.	2	Set token PID checksum error 2'b00: Not Set 2'b01: Normal Token PID checksum error is set 2'b10: SPLIT Token PID checksum error is set
3.	TKN_WRONG_DEV_ADDR	Set device address to incorrect value.	7	Wrong Device Address
4.	TKN_WRONG_EP_NUM	Set endpoint number to incorrect value.	4	Wrong Endpoint Number
5.	TKN_CRC5_ERR	Set CRC5 error in token packet.	2	Set Token CRC5 Error 2'b00: Nor Set 2'b01: Normal Token CRC5 Error is Set. 2'b10: SPLIT Token CRC5 Error is Set
6.	TKN_WRONG_HUB_ADDR	Set Wrong HUB address.	7	Wrong HUB Address
7.	TKN_WRONG_PORT_NO	Set Wrong Port Number.	7	Wrong Port Number
8.	TKN_WRONG_SPLIT_EP_TYPE	Set Wrong Endpoint type.	2	2'b00: USB_CONTROL 2'b01: USB_ISO 2'b10: USB_BULK 2'b11: USB_INT
9.	TKN_SKIP	Skip the Normal Token, SSPLIT Token and SPLIT Token.	2	2'b0: Not to Skip 2'b1: Skip Normal Token 2'b2: Skip SSPLIT Token 2'b3: Skip CSPLIT Token
Data Phase				
10.	DATA_INVLD_PID	Set Invalid PID in Split Token Phase.	4	Wrong Token PID

11.	DATA_INVLD_PID	Set invalid PID value for data packet.	4	Wrong Data PID
12.	DATA_PID_CHKSUM_ERR	Set PID checksum error in data packet.	1	Set data PID checksum error 0: Not Set 1: Set
13.	DATA_INVLD_PKT_LEN	Set invalid packet length for data packet.	32	Invalid data packet length
14.	DATA_CRC16_ERR	Set crc16 error in data packet to be sent.	1	Set data CRC16 Error 0: Not Set 1: Set
15.	DATA_SKIP	Instructs host BFM to skip data stage during SETUP or OUT transaction.	1	Set data stage skip 0: No Skip 1: Skip
Handshake Phase (For In Transfer)				
16.	HSK_INVLD_PID	Set invalid PID value for handshake packet to be sent.	4	Wrong Handshake PID
17.	HSK_PID_CHKSUM_ERR	Set PID checksum error in handshake packet.	1	Set handshake PID checksum error 0: Not Set 1: Set
18.	HSK_SKIP	Instructs Host BFM to skip handshake stage.	1	Skip Handshake Stage 0: No Skip 1: Skip
19.	HSK_INVLD_PKT_LEN	Set invalid handshake packet length for handshake packet.	1	Set invalid handshake packet length 0: Not Set 1: Set
Control Endpoint Errors				
20.	INVLD_SETUP_PKT_LEN	Set invalid setup packet length.	1	Set invalid setup packet length 0: Not Set 1: Set

21.	SKIP_SETUP_DATA	No setup data packet after setup token.	1	Skip setup data packet 0: No skip 1: Skip
22.	DISABLE_KJKJKJ	Disable kjkjkj chirp sequence after receiving chirp K from the device.	1	1: Disable 0: Not disable

Examples:

1. Wrong token PID for setup transaction

```
inject_err (`TKN_INVLD_PID, `USB_IN_PID);
```

2. No data stage after setup stage

```
inject_Err (`DATA_SKIP, 1);
```

3. To skip the SSPLIT Token from the transaction

```
inject_err (`TKN_SKIP, 2'b2)
```

host_status_rd

This task is used to read different values of the host BFM model to design few conditional based test cases at the top level. User can jump at different level depending upon the read values. [Table 4-10.](#) shows task argument details.

Syntax:

```
host_status_rd (Parameter, Parameter output value);
```

Table 4-10. host_status_rd

No.	Parameter	Description	Parameter Output Value
1.	VBUS_STATUS	Specifies the status of vbus.	0: VBUS is disabled 1: VBUS is enabled
2.	DEVICE_CONNECTION	Specifies whether device is connected or not.	0: Device is not connected 1: Device is connected
3.	HOST_SPEED	Specifies the speed of operation.	2'b00: High Speed 2'b01: Full Speed 2'b10: Low Speed
4.	DEVICE_SUSPEND	Specifies whether attached device is in suspend state or not.	0: Device is not in suspend state 1: Device is in suspend state
5.	LINE_STATE	Specifies the value of line-state.	-

6.	HOST_IDLE	Specifies whether host is performing any command request or is in idle state.	0: Busy 1: Idle
7.	LAST_TRANS_STATUS	Specifies the status of the last transaction.	8'h1: ACK_SENT 8'h2: TIMEOUT 8'h3: CRC16_RCVD 8'h4: ACK_RCVD 8'h5: NAK_RCVD 8'h6: STALL_RCVD 8'h7: NYET_RCVD 8'h8: ERR_RCVD 8'h9: DATA_RCVD

Examples:

1. Check status of Device Speed

```
host_status_rd (`HOST_SPEED, status);
```



Status is some local variable of module whose value determines device speed.

2. Check status of last transaction

```
host_status_rd (`LAST_TRANS_STATUS,  
last_trans_status);
```



last_trans_status is some local variable of module whose value determines last transaction status.

This section describes the list of events generated by host BFM.

Table 5-1. shows the events and description.

Table 5-1. Events of BFM	
Event	Description
Vbus/Device Connection Events	
event_vbus_on	Vbus supply is enabled.
event_vbus_off	Vbus supply is disabled.
event_dev_conn	Device connection detected.
event_dev_disconn	Device disconnection detected.
Reset Events	
event_reset_end	Completion of USB host initiated reset.
event_host_bfm_reset_done	Completion of Host BFM reset.
Transaction Events	
event_in_txn_cmplt	Completion of IN transaction
event_out_txn_cmplt	Completion of OUT transaction
event_setup_txn_cmplt	Completion of SETUP transaction
event_ping_txn_cmplt	Completion of PING transaction
event_sof_txn_cmplt	Completion of SOF transaction
Suspend/Resume Events	
event_suspend_state	Device goes into suspend state
event_resume_state	Device resumes back from the suspend state
Handshake Events	
event_ack_rcvd	ACK handshake received

event_nak_rcvd	NAK handshake received
event_nyet_rcvd	NYET handshake received
event_stall_rcvd	STALL handshake received
event_err_rcvd	ERR handshake received
Data PID Receive Events	
event_data0_rcvd	Data packet with DATA0 PID received
event_data1_rcvd	Data packet with DATA1 PID received
event_data2_rcvd	Data packet with DATA2 PID received
event_mdata_rcvd	Data packet with MDATA PID received
Frame Related Events	
event_new_micro_frame	New micro frame received.
event_new_frame	New frame received.

USB 2.0 Monitor monitors USB 2.0 D+ and D- line and prints USB Transaction into log file in user friendly manner. It generates monitor_display.htm log file with various colors so user can easily decode all data in each transaction.

Log File Header Field Description

Figure 6-1. shows Sample of Log File Header.

Figure 6-1. Sample of Log File Header

SIM TIME	UNIT	SPEED	DIR	PKT TYPE	PID TYPE	DATA BYTES [LSB : MSB]	CRC16 BYTES [LSB : MSB]	PKT LEN
----------	------	-------	-----	----------	----------	------------------------	-------------------------	---------

Table 6-1. describes the log file header column description.

Name	Description
SIM TIME	Simulation Time Value
UNIT	Simulation Time Unit
SPEED	Speed of Transaction
DIR	Direction of Transaction
PKT TYPE	Packet Type
PID TYPE	PID Type
DATA BYTES [LSB:MSB]	Data Bytes of Packet (From LSB to MSB)
CRC16 BYTES [LSB:MSB]	CRC16 Bytes (From LSB to MSB)
PKT LEN	Length of packet in Bytes

Example:

USB Transaction that monitor prints in log file as shown in [Figure 6-2](#).
Get Device Descriptor with 8 bytes request is mention in below example.

Color coding for monitor_display.htm file:

- **GREEN:** Host to Device Transaction
- **BLUE:** Device to Host Transaction
- **RED:** Error Injected Data

Figure 6-2. USB Transaction

```

| | | | P | P | | | | | P |
| | S | I | K | I | | | | | K |
| U | P | D | T | D | | | | | T |
| N | E | I | I | | | | | |
SIM TIME | I | E | R | T | T | DATA BYTES [LSB : MSB] | CRC16 BYTES |
| T | D | Y | Y | | | | [LSB : MSB] | L |
| | | P | P | | | | | E |
| | | E | E | | | | | N |
-----
Device Enumeration Starts
-----
1. Get Device Descriptor Request (8-bytes)
-----
| 123479000 ps | HS | H | TOKEN | Setup | Add 0x00 | Ep No 0x0 | CRC5 0x02 | | | 3 |
| 123927000 ps | HS | H | DATA | Data0 | 0x80 0x06 0x00 0x01 0x00 0x00 0x08 0x00 | | 0xeb 0x94 | 11 |
| 124359000 ps | HS | D | HSK | ACK | | | | | 1 |
-----
| 124855000 ps | HS | H | TOKEN | IN | Add 0x00 | Ep No 0x0 | CRC5 0x02 | | | 3 |
| 125207000 ps | HS | D | DATA | Data1 | 0x12 0x01 0x00 0x02 0xff 0x00 0xff 0x40 | | 0x26 0x35 | 11 |
| 125751000 ps | HS | H | HSK | ACK | | | | | 1 |
-----
| 126279000 ps | HS | H | TOKEN | OUT | Add 0x00 | Ep No 0x0 | CRC5 0x02 | | | 3 |
| 126727000 ps | HS | H | DATA | Data1 | | | | | 0 |
| 127047000 ps | HS | D | HSK | ACK | | | | | 1 |
-----

```

[Figure 6-3](#). shows sample of Error Injected data. In this test we give Control Endpoint number 0xF instead of 0x0 in Setup stage. So Endpoint no (with Error Injection) is printed in RED color.

Figure 6-3. Example of Error Injected data

```

-----
# Test Index:1
Control Endpoint Test : Invalid endpoint number in setup stage

| 188967000 ps | HS | H | TOKEN | Setup | Add 0x02 | Ep No 0xf | CRC5 0x18 | | | 3 |
| 189415000 ps | HS | H | DATA | Data0 | 0x80 0x06 0x00 0x01 0x00 0x00 0x08 0x00 | | 0xeb 0x94 | 11 |
-----
| 205751000 ps | HS | H | TOKEN | Setup | Add 0x02 | Ep No 0xf | CRC5 0x18 | | | 3 |
| 206199000 ps | HS | H | DATA | Data0 | 0x80 0x06 0x00 0x01 0x00 0x00 0x08 0x00 | | 0xeb 0x94 | 11 |
-----
| 222535000 ps | HS | H | TOKEN | Setup | Add 0x02 | Ep No 0xf | CRC5 0x18 | | | 3 |
| 222983000 ps | HS | H | DATA | Data0 | 0x80 0x06 0x00 0x01 0x00 0x00 0x08 0x00 | | 0xeb 0x94 | 11 |
-----
| 226583000 ps | HS | H | TOKEN | SOF | Frm No 0x000 | | CRC5 0x02 | | | 3 |
| 239319000 ps | HS | H | TOKEN | Setup | Add 0x02 | Ep No 0xf | CRC5 0x18 | | | 3 |
| 239767000 ps | HS | H | DATA | Data0 | 0x80 0x06 0x00 0x01 0x00 0x00 0x08 0x00 | | 0xeb 0x94 | 11 |
-----

```

Log File Summary

At the end of simulation following performance parameters will display based on simulation.

Table 6-2 describes the performance parameters.

Table 6-2. Performance Parameters	
Commands	No of occurrence
IN Commands	Number of IN Commands
OUT Commands	Number of OUT Commands
SOF Commands	Number of SOF Commands
SETUP Commands	Number of SETUP Commands
DATA0 Commands	Number of DATA0 Commands
DATA1 Commands	Number of DATA1 Commands
DATA2 Commands	Number of DATA2 Commands
MDATA Commands	Number of MDATA Commands
ACK Commands	Number of ACK Commands
NACK Commands	Number of NACK Commands
STALL Commands	Number of STALL Commands

NYET Commands	Number of NYET Commands
ERROR Commands	Number of ERROR Commands
SPLIT Commands	Number of SPLIT Commands
PING Commands	Number of PING Commands

Figure 6-4. Performance Parameter

Performance Parameter	
Commands	No of occurrence
IN Commands	73
OUT Commands	98
SOF Commands	30
SETUP Commands	62
DATA0 Commands	108
DATA1 Commands	82
DATA2 Commands	1
MDATA Commands	0
ACK Commands	79
NACK Commands	24
STALL Commands	0
NYET Commands	1
ERROR Commands	0
SPLIT Commands	0
PING Commands	0

This section describes list of tasks used in the USB 2.0 Device Verification environment. [Table A-1](#) shows the tasks details.

Table: A-1. Tasks of USB2.0 Device Verification Environment		
No.	Task Name	Description
Simulation Test Suite Tasks		
1.	initial_seq_task	This task configures Host BFM different timers and set features, also it contains task to configure Host BFM to supply VBUS and perform reset sequence on the attached device.
2.	normal_enumeration_task	This task configures Host BFM to perform enumeration sequence on High Speed Device.
3.	fs_enumeration_task	This task configures Host BFM to perform enumeration sequence on Full Speed Device.
4.	Control_Ep_Tests	This task performs different control operations to check the functionality of control endpoint in high speed device.
5.	Bulk_In_Tests	This task performs different bulk in operations to check the functionality of bulk in endpoint in high speed device.
6.	Bulk_Out_Tests	This task performs different bulk out operations to check the functionality of bulk out endpoint in high speed device.
7.	Interrupt_In_Tests	This task performs different interrupt IN operations to check the functionality of interrupt in endpoint in high speed device.
8.	Control_Ep_Split_Tests	This task performs different control operations to check the functionality of control endpoint in full speed device.
9.	Bulk_In_Split_Tests	This task performs different bulk IN operations to check the functionality of bulk in endpoint in full speed device.
10.	Bulk_Out_Split_Tests	This task performs different bulk OUT operations to check the functionality of bulk out endpoint in full speed device.
11.	Interrupt_In_Split_Tests	This task performs different interrupt IN operations to check the functionality of interrupt in endpoint in full speed device.
12.	Port_Config	This task performs different requests on HUB to configure its port.
13.	Hub_Enumeration	This task enumerates the HUB.

initial_seq_task

This task configures Host BFM different timers, set features, it contains task to configure Host BFM to supply VBUS and perform reset sequence on the attached device.

Syntax:

```
initial_seq_task
```



This task is used in *initialization_tasks.v* file.

normal_enumeration_tasks

This task configures Host BFM to perform enumeration sequence on DUT.

Syntax:

```
normal_enumeration_task
```



This task is used in *enumeration_tasks.v* file.

fs_enumeration_task

This task configures Host BFM to perform enumeration sequence on full speed / low speed device.

Syntax:

```
fs_enumeration_task
```



This task is used in *enumeration_task.v* file.

Control_Ep_Tests

This task performs different control operations to check the functionality of control endpoint.

Syntax:

```
Control_Ep_Tests
```



This task is used in *control_ep_tests.v* file.

Bulk_In_Tests

This task performs different bulk in operations to check the functionality of bulk in endpoint. [Table A-2](#) shows the argument details.

Syntax:

```
Bulk_In_Tests (Device Address, Endpoint No, Packet  
Size);
```

Table: A-2. Bulk_In_Tests

No.	Argument	Size (Bits)	Description
1.	Device Address	7	Device address indicates with whom host to communicate.
2.	Endpoint No	4	Device endpoint number Indicates for data communication.
3.	Packet Size	12	Packet size indicates the size of packet in bytes for different operations to be performed.

Example:

If device address = 7'h01, Endpoint no for bulk in = 4'h00 and packet size = 512 bytes then,

```
Bulk_In_Tests (7'h01, 4'h00,12'h200);
```



This task is used in *bulk_in_tasks.v* file.

Bulk_Out_Tests

This task performs different bulk out operations to check the functionality of bulk in endpoint. [Table A-3](#) shows the argument details.

Syntax:

```
Bulk_Out_Tests (Device Address, Endpoint No, Packet  
Size);
```

Table: A-3. Bulk_Out_Tests

No.	Argument	Size (Bits)	Description
1.	Device Address	7	Device address indicates with whom host to communicate.
2.	Endpoint No	4	Device endpoint number Indicates for data communication.
3.	Packet Size	12	Packet size indicates the size of packet in bytes for different operations to be performed.

Example:

If device address = 7'h01, Endpoint no for bulk out = 4'h01 and packet size = 512 bytes then,

```
Bulk_Out_Tests (7'h01, 4'h01,12'h200);
```



This task is used in *bulk_out_tasks.v* file.

Interrupt_In_Tests

This task performs different interrupt IN operations to check the functionality of interrupt in endpoint. [Table A-4](#) shows the Interrupt_In_Tests details.

Syntax:

```
Interrupt_In_Tests (Device Address, Endpoint No, Packet
Size);
```

Table: A-4. Interrupt_In_Tests

No.	Argument	Size (Bits)	Description
1.	Device Address	7	Device address indicates with whom host to communicate.
2.	Endpoint No	4	Device endpoint number Indicates for data communication.
3.	Packet Size	12	Packet size indicates the size of packet in bytes for different operations to be performed.



This task is used in *interrupt_in_tasks.v* file.

Control_Ep_Split_Tests

This task performs different control operations to check the functionality of control endpoint.

Syntax:

```
Control_Ep_Split_Tests
```



This task is used in *control_ep_split_tests.v* file.

Bulk_In_Split_Tests

This task performs different bulk IN operations to check the functionality of bulk in endpoint. [Table A-5](#) shows the Bulk_In_Split_Tests details.

Syntax:

```
Bulk_In_Split_Tests (Device Address, Endpoint No,  
Packet Size);
```

Table: A-5. Bulk_In_Split_Tests

No.	Argument	Size (Bits)	Description
1.	Device Address	7	Device address indicates with whom host to communicate.
2.	Endpoint No	4	Device endpoint number Indicates for data communication.
3.	Packet Size	12	Packet size indicates the size of packet in bytes for different operations to be performed.



This task is used in *bulk_in_split_tasks.v* file.

Bulk_Out_Split_Tests

This task performs different bulk OUT operations to check the functionality of bulk out endpoint. [Table A-6](#) shows the Bulk_Out_Split_Tests details.

Syntax:

```
Bulk_Out_Split_Tests (Device Address, Endpoint No,  
Packet Size);
```

Table: A-6. Bulk_Out_Split_Tests

No.	Argument	Size (Bits)	Description
1.	Device Address	7	Device address indicates with whom host to communicate.
2.	Endpoint No	4	Device endpoint number Indicates for data communication.
3.	Packet Size	12	Packet size indicates the size of packet in bytes for different operations to be performed.



This task is used in *bulk_out_split_tasks.v* file.

Interrupt_In_Split_Tests

This task performs different interrupt IN operations to check the functionality of interrupt in endpoint. [Table A-7](#) shows the Interrupt_In_Split_Tests details.

Syntax: `Interrupt_In_Split_Tests (Device Address, Endpoint No, Packet Size);`

Table: A-7. Interrupt_In_Split_Tests

No.	Argument	Size (Bits)	Description
1.	Device Address	7	Device address indicates with whom host to communicate.
2.	Endpoint No	4	Device endpoint number Indicates for data communication.
3.	Packet Size	12	Packet size indicates the size of packet in bytes for different operations to be performed.



This task is used in *interrupt_in_split_tasks.v* file.

Port_Config

This task performs different requests on HUB to configure its port.

[Table A-8](#) shows the Port_Config details.

Syntax: `Port_Config`

Table: A-8. Port_Config

No.	Argument	Size (Bits)	Description
1.	Port Number	7	It indicates port number to configure for data communication.



This task is used in *port_config.v* file.

Defined Files

Hub_Enumeration

This task enumerates the HUB.

Syntax: `Hub_Enumeration`



- This task is used in *hub_enumeration.v* file.
- User defined tasks can be modified by user as per requirements.

User must include three defines files named “*user_defines.v*”, “*usb_spec_defines.v*” and “*defines.v*”. In which “*user_defines.v*” and “*usb_spec_defines.v*” are user defined files and later one is internal file. User

can modified “*user_defines.v*” and “*usb_spec_defines.v*” according to requirement.



User must not change these two files names otherwise VIP would not identify these two files during simulation.

This section describes different examples for the USB 2.0 Device Verification environment.

Example 1:

This example shows the basic initialization of host BFM to enable VBUS supply and reset connected device.



User should set other parameters in `cfg_host` task accordingly and set different timer values using `cfg_timer` before starting any sequence.

```
do_cmd (`HOST_BFM_RESET);
@ (`event_host_bfm_reset_done);
$display("# Host BFM Reset Completed\n");

// Configure Host BFM timers
configure_hbfm_timers;
$display("# Timers are configured\n");

// Configure Host BFM features
configure_hbfm_features;
$display("# Features are set\n");

// Enable Vbus supply
`do_cmd (`VBUS_ENABLE);

// Wait until Vbus supply is turned on
@ (`event_vbus_on );
$display("vbus is enabled\n");

// Wait until device is connected
@ (`event_dev_conn);
$display("Device is connected\n");

wait_for_attach_stable;
```

```

// Reset connected device
`do_cmd (`USB_RESET);

// Wait until reset operation finishes.
@ (`event_reset_end);

```

Example 2:

This example shows to configure Host BFM to perform GET DEVICE DESCRIPTOR request with High Speed Device in default state.

```

// Setup Stage
cfg_token (`TOKEN_PID, `USB_SETUP_PID);
cfg_token (`DEV_ADDRESS, `DEFAULT_ADDR);
cfg_token (`EP_NUMBER, `DEFAULT_EP);
cfg_setup_data (`SETUP_REQ_DIR, `USB_DEVICE_TO_HOST);
cfg_setup_data (`SETUP_REQ_TYPE, `USB_TYPE_STANDARD);
cfg_setup_data (`SETUP_REQ_RECIPIENT, `USB_REC_DEV);
cfg_setup_data (`SETUP_bREQUEST, `GET_DESCRIPTOR);
cfg_setup_data (`SETUP_wVALUE, {DESC_TYPE_DEVICE, 8'h0});
cfg_setup_data (`SETUP_wINDEX, 16'h0);
cfg_setup_data (`SETUP_wLENGTH, 16'h8);
do_cmd (`USB_SETUP);
@ (`event_setup_txn_cmplt);

// Data Stage
cfg_token (`TOKEN_PID, `USB_IN_PID);
cfg_token (`DEV_ADDRESS, `DEFAULT_ADDR);
cfg_token (`EP_NUMBER, `DEFAULT_EP);
do_cmd (`USB_IN);
@ (`event_in_txn_cmplt);

// Status Stage
cfg_token (`TOKEN_PID, `USB_OUT_PID);
cfg_token (`DEV_ADDRESS, `DEFAULT_ADDR);
cfg_token (`EP_NUMBER, `DEFAULT_EP);
cfg_data_pkt (`TRANSFER_TYPE, `USB_CONTROL);
cfg_data_pkt (`DATA_PID, `USB_DATA0_PID);
cfg_data_pkt (`PKT_SIZE, `ZERO_LENGTH);

```

```
do_cmd (`USB_OUT);
@ (`event_out_txn_cmplt);
```

Example 3:

This example shows how to configure Host BFM to perform SET ADDRESS request with High Speed Device in default state.

```
//Setup Stage
cfg_token (`TOKEN_PID, `USB_SETUP_PID);
cfg_token (`DEV_ADDRESS, `DEFAULT_ADDR);
cfg_token (`EP_NUMBER, `DEFAULT_EP);

cfg_setup_data (`SETUP_REQ_DIR, `USB_HOST_TO_DEVICE);
cfg_setup_data (`SETUP_REQ_TYPE, `USB_TYPE_STANDARD);
cfg_setup_data (`SETUP_REQ_RECIPIENT, `USB_REC_DEV);
cfg_setup_data (`SETUP_bREQUEST, `SET_ADDRESS);
cfg_setup_data (`SETUP_wVALUE, 1); // Address to be set
cfg_setup_data (`SETUP_wINDEX, 16'h0);
cfg_setup_data (`SETUP_wLENGTH, 16'h0);
do_cmd (`USB_SETUP);
@ (`event_setup_txn_cmplt);

// Status Stage
cfg_token (`TOKEN_PID, `USB_IN_PID);
cfg_token (`DEV_ADDRESS, `DEFAULT_ADDR);
cfg_token (`EP_NUMBER, `DEFAULT_EP);
do_cmd (`USB_IN);
@ (`event_in_txn_cmplt);
```

Example 4:

This example shows to configure Host BFM to perform GET DEVICE DESCRIPTOR request with Full Speed device in default state.

```
// Setup Stage
`cfg_token (`TOKEN_PID, `USB_SETUP_PID);
`cfg_token (`DEV_ADDRESS, `DEFAULT_ADDR);
`cfg_token (`EP_NUMBER, `DEFAULT_EP);
`cfg_token (`SPLIT_HUB_ADDRESS, `HUB_ADDRESS);
`cfg_token (`SPLIT_START_COMPLETE, `START_SPLIT);
`cfg_token (`SPLIT_HUB_PORT_NUMBER, `FS_PORT_NO);
`cfg_token (`SPLIT_SPEED, `USB_FS);
```

```
`cfg_token (`SPLIT_EP_TYPE, `USB_CONTROL);
`cfg_setup_data (`SETUP_REQ_DIR, `USB_DEVICE_TO_HOST);
`cfg_setup_data (`SETUP_REQ_TYPE, `USB_TYPE_STANDARD);
`cfg_setup_data (`SETUP_REQ_RECIPIENT, `USB_REC_DEVICE);
`cfg_setup_data (`SETUP_bREQUEST, `GET_DESCRIPTOR);
`cfg_setup_data (`SETUP_wVALUE, {`DESC_TYPE_DEVICE,8'h0});
`cfg_setup_data (`SETUP_wINDEX, 16'h0);
`cfg_setup_data (`SETUP_wLENGTH, 16'h8);
`do_cmd (`USB_SPLIT_SETUP);
@ (`event_setup_txn_cmplt);

// Data Stage
`cfg_token (`TOKEN_PID, `USB_IN_PID);
`cfg_token (`DEV_ADDRESS, `DEFAULT_ADDR);
`cfg_token (`EP_NUMBER, `DEFAULT_EP);
`cfg_data_pkt (`TRANSFER_TYPE, `USB_CONTROL);
`cfg_data_pkt (`DATA_PID, `USB_DATA1_PID);
`cfg_data_pkt (`PKT_SIZE, 16'h12);
`do_cmd (`USB_SPLIT_IN);
@ (`event_in_txn_cmplt);

// Status Stage
`cfg_token (`TOKEN_PID, `USB_OUT_PID);
`cfg_token (`DEV_ADDRESS, `DEFAULT_ADDR);
`cfg_token (`EP_NUMBER, `DEFAULT_EP);
`cfg_data_pkt (`TRANSFER_TYPE, `USB_CONTROL);
`cfg_data_pkt (`DATA_PID, `USB_DATA1_PID);
`cfg_data_pkt (`PKT_SIZE, `ZERO_LENGTH);
`do_cmd (`USB_SPLIT_OUT);
@ (`event_out_txn_cmplt);
```



Appendix C: Sample Monitor Log File

Example C-1. Sample Monitor Log File

```

.....
# USB 2.0 HOST BFM Monitor #
.....

Color Coding:
GREEN:- Host to Device Transaction
BLUE :- Device to Host Transaction
RED  :- Error Injected Data

-----
|                               |
|                               |
|-----|-----|
| Name           | Description           |
|-----|-----|
| SIM TIME       | Simulation Time Value |
| UNIT           | Simulation Time Unit  |
| SPEED          | Speed Of Transaction  |
| DIR            | Direction Of Transaction |
| PKT TYPE       | Packet Type           |
| PID TYPE       | PID Type              |
| DATA BYTES (LSB : MSB) | Data Bytes Of Packet (From LSB to MSB) |
| CRC16 BYTES (LSB : MSB) | CRC16 Bytes (From LSB to MSB) |
| PKT LEN        | Length Of Packet In Bytes |
|-----|-----|

| | | | | P | P | | | | P | | | |
| | | | | K | I | | | | K |
| | | | | U | P | D | T | D | | | | T |
| | | | | N | E | I | | | | | | |
| SIM TIME | I | E | R | T | T | | DATA BYTES (LSB : MSB) | |
| | | | | T | D | | Y | Y | | | | |
| | | | | P | P | | | | | |
| | | | | I | E | I | | | | | |

-----

Device Enumeration Starts

-----

1. Get Device Descriptor Request (8-bytes)

| 123479000 ps | HS | H | TOKEN | Setup | Add 0x00 | Ep No 0x0 | CRC5 0x02 | | | 3 |
| 123927000 ps | HS | H | DATA | Data0 | 0x90 0x06 0x00 0x01 0x00 0x00 0x08 0x00 | | | 11 |
| 124359000 ps | HS | D | HSK | ACK | | | | | 1 | |
|---|---|---|---|---|---|---|---|---|---|---|
| 124855000 ps | HS | H | TOKEN | IN | Add 0x00 | Ep No 0x0 | CRC5 0x02 | | | 3 |
| 125207000 ps | HS | D | DATA | Data1 | 0x12 0x01 0x00 0x02 0xff 0x00 0xff 0x40 | | | 11 |
| 125751000 ps | HS | H | HSK | ACK | | | | | 1 |

```

```

| 126279000 ps | HS | H | TOKEN | OUT | Add 0x00 | Ep Mo 0x0 | CRC5 0x02 | | | 3 |
| 126727000 ps | HS | H | DATA | Data1 | | 0x00 0x00 | | 0 |
| 127047000 ps | HS | D | HSK | ACK | | | | 1 |
| -----
2. Set Address Request
| 128343000 ps | HS | H | TOKEN | Setup | Add 0x00 | Ep Mo 0x0 | CRC5 0x02 | | | 3 |
| 128791000 ps | HS | H | DATA | Data0 | 0x00 0x05 0x02 0x00 0x00 0x00 0x00 0x00 | | 0x0b 0x16 | 11 |
| 129223000 ps | HS | D | HSK | ACK | | | | 1 | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 129719000 ps | HS | H | TOKEN | IN | Add 0x00 | Ep Mo 0x0 | CRC5 0x02 | | | 3 |
| 130023000 ps | HS | D | DATA | Data1 | | 0x00 0x00 | | 0 |
| 130439000 ps | HS | H | HSK | ACK | | | | 1 |
| -----
3. Get Device Descriptor Request ( Full )
| 131767000 ps | HS | H | TOKEN | Setup | Add 0x02 | Ep Mo 0x0 | CRC5 0x15 | | | 3 |
| 132218000 ps | HS | H | DATA | Data0 | 0x00 0x06 0x00 0x01 0x00 0x00 0x12 0x00 | | 0xe0 0xf4 | 11 |
| 132647000 ps | HS | D | HSK | ACK | | | | 1 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 133143000 ps | HS | H | TOKEN | IN | Add 0x02 | Ep Mo 0x0 | CRC5 0x15 | | | 3 |
| 133495000 ps | HS | D | DATA | Data1 | 0x12 0x01 0x00 0x02 0xff 0x00 0xff 0x40 0x72 0x17 0x02 0x00 0x10 0x00 0x01 0x02 | | | |
| 133863000 ps | | | | | | | | 0x03 0x01 | | 0xaf 0x88 | 21 |
| 134199000 ps | HS | H | HSK | ACK | | | | 1 | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 134727000 ps | HS | H | TOKEN | OUT | Add 0x02 | Ep Mo 0x0 | CRC5 0x15 | | | 3 |
| 135175000 ps | HS | H | DATA | Data1 | | 0x00 0x00 | | 0 |
| 135495000 ps | HS | D | HSK | ACK | | | | 1 |
| -----
4. Get Only Configuration Descriptor Request (9-bytes)
| 136791000 ps | HS | H | TOKEN | Setup | Add 0x02 | Ep Mo 0x0 | CRC5 0x15 | | | 3 |
| 137239000 ps | HS | H | DATA | Data0 | 0x00 0x06 0x00 0x02 0x00 0x00 0x09 0x00 | | 0xae 0x04 | 11 |
| 137671000 ps | HS | D | HSK | ACK | | | | 1 |

```

```

|-----|
| 138167000 ps | HS | H | TOKEN | IN | Add 0x02 | Ep Mo 0x0 | CRC5 0x15 | | | 3 |
| 138519000 ps | HS | D | DATA | Data1 | 0x09 0x02 0x4a 0x00 0x01 0x01 0x04 0xc0 0x32 | | 0x59 0x6a | 12 |
| 139079000 ps | HS | H | HSK | ACK | | | | | 1 | |
|---|---|---|---|---|---|---|---|---|---|---|
| 139607000 ps | HS | H | TOKEN | OUT | Add 0x02 | Ep Mo 0x0 | CRC5 0x15 | | | 3 |
| 140055000 ps | HS | H | DATA | Data1 | | | | | 0 |
| 140375000 ps | HS | D | HSK | ACK | | | | | 1 |
|-----|
|-----|
| 5. Get Full Configuration Descriptor Request
|-----|
| 141671000 ps | HS | H | TOKEN | Setup | Add 0x02 | Ep Mo 0x0 | CRC5 0x15 | | | 3 |
| 142119000 ps | HS | H | DATA | Data0 | 0x80 0x06 0x00 0x02 0x00 0x00 0x20 0x00 | | 0xb1 0x94 | 11 |
| 142551000 ps | HS | D | HSK | ACK | | | | | 1 | |
|---|---|---|---|---|---|---|---|---|---|---|
| 143047000 ps | HS | H | TOKEN | IN | Add 0x02 | Ep Mo 0x0 | CRC5 0x15 | | | 3 |
| 143399000 ps | HS | D | DATA | Data1 | 0x09 0x02 0x4a 0x00 0x01 0x01 0x04 0xc0 0x32 0x09 0x04 0x00 0x08 0xff 0x00 | | | |
| 143891000 ps | |-----| 0xff 0x05 0x07 0x05 0x81 0x02 0x00 0x02 0x01 0x07 0x05 0x02 0x00 0x02 0x01 | | 0xb3 0x49 | 35 |
| 144327000 ps | HS | H | HSK | ACK | | | | | 1 | |
|---|---|---|---|---|---|---|---|---|---|---|
| 144855000 ps | HS | H | TOKEN | OUT | Add 0x02 | Ep Mo 0x0 | CRC5 0x15 | | | 3 |
| 145303000 ps | HS | H | DATA | Data1 | | | | | 0 |
| 145623000 ps | HS | D | HSK | ACK | | | | | 1 |
|-----|
|-----|
| 6. Set Configuration Request
|-----|
| 146919000 ps | HS | H | TOKEN | Setup | Add 0x02 | Ep Mo 0x0 | CRC5 0x15 | | | 3 |
| 147367000 ps | HS | H | DATA | Data0 | 0x00 0x09 0x01 0x00 0x00 0x00 0x00 0x00 | | 0x27 0x25 | 11 |
| 147799000 ps | HS | D | HSK | ACK | | | | | 1 | |
|---|---|---|---|---|---|---|---|---|---|---|
| 148295000 ps | HS | H | TOKEN | IN | Add 0x02 | Ep Mo 0x0 | CRC5 0x15 | | | 3 |
| 148599000 ps | HS | D | DATA | Data1 | | | | | 0 |
| 149015000 ps | HS | H | HSK | ACK | | | | | 1 |
|-----|

```

```

.....
Bulk In Tests ( Max Packet Size = 64-bytes )
.....
# Test Index:1

Bulk In Test : Normal operation with its max pkt size

| 151415000 ps | HS | H | TOKEN | IN | Add 0x02 | Ep No 0x1 | CRC5 0x03 | | | 3 |
|
| 151767000 ps | HS | D | DATA | Data0 | 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f | | | |
| 152871000 ps | | ---- | 0x10 0x11 0x12 0x13 0x14 0x15 0x16 0x17 0x18 0x19 0x1a 0x1b 0x1c 0x1d 0x1e 0x1f | | | |
| 152871000 ps | | ---- | 0x20 0x21 0x22 0x23 0x24 0x25 0x26 0x27 0x28 0x29 0x2a 0x2b 0x2c 0x2d 0x2e 0x2f | | | |
| 152871000 ps | | ---- | 0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x3a 0x3b 0x3c 0x3d 0x3e 0x3f | 0x26 0xf7 | 67 |
|
| 153207000 ps | HS | H | HSK | ACK | | | | | 1 |
|
.....

Bulk Out Test ( Max Packet Size = 64-bytes )
.....
# Test Index:2

Bulk Out Test : Normal operation with its max pkt size

| 154535000 ps | HS | H | TOKEN | OUT | Add 0x02 | Ep No 0x2 | CRC5 0x10 | | | 3 |
|
| 154983000 ps | HS | H | DATA | Data0 | 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f | | | |
| 156087000 ps | | ---- | 0x10 0x11 0x12 0x13 0x14 0x15 0x16 0x17 0x18 0x19 0x1a 0x1b 0x1c 0x1d 0x1e 0x1f | | | |
| 156087000 ps | | ---- | 0x20 0x21 0x22 0x23 0x24 0x25 0x26 0x27 0x28 0x29 0x2a 0x2b 0x2c 0x2d 0x2e 0x2f | | | |
| 156087000 ps | | ---- | 0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x3a 0x3b 0x3c 0x3d 0x3e 0x3f | 0x26 0xf7 | 67 |
|
| 156327000 ps | HS | D | HSK | ACK | | | | | 1 |
|
.....

|-----|-----|
| Performance Parameter |
|-----|-----|
| Commands | No of occurrence |
|-----|-----|
| IN Commands | 7 |
| OUT Commands | 5 |
| SOF Commands | 0 |
| SETUP Commands | 6 |
| DATA0 Commands | 8 |
| DATA1 Commands | 10 |
| DATA2 Commands | 0 |
| MDATA Commands | 0 |
| ACK Commands | 18 |
| NACK Commands | 0 |
| STALL Commands | 0 |
| NYET Commands | 0 |
| ERROR Commands | 0 |
| SPLIT Commands | 0 |
| PING Commands | 0 |
|-----|-----|

```