UART2BUS

Verificaiton Plan *Opencores.com*

Release in 26. June 2017

Hany Salah
VLSI Verification Engineer

Table of Contents

Illustration Index

Index of Tables

1 About the Document

1.1 Description

This document describes the verification plan used to verify UART2BUS Open-cores project
Released by Moti Litochevski

1.2 References

- Bruce Wile, John Goss, Wolfgang Roesner – Comprehensive Functional Verification – The Complete Industry Cycle – Systems on Silicon (2005).
- System Verilog For Verification – a guide to learning the test-bench language features by Chris Spear Gregory J Tumbash (2012).
- A Practical Guide to Adopting the Universal Verification Methodology (UVM) by Sharon Rosenberg, Kathleen Meade – Cadence Design Systems (2010).

1.3 Log Details

Table 1: Log details

Version	Date	Editor	Description
1	December 23, 2015	Hany Salah	• Document creation • Add Design Entity
2	December 24, 2015	Hany Salah	• Add Design Specifications • Add System Behavioral Description • Modify Design Entity
3	December 25, 2015	Hany Salah	• Improve System Behavioral Description • Modify Design Specifications
4	December 29, 2015	Hany Salah	• Modify System Behavioral Description. • Create Test Plan.
5	December 30, 2015	Hany Salah	• Improve Test Plan (UART Features).
6	December 31, 2015	Hany Salah	• Add Transaction Content.
7	January 01, 2016	Hany Salah	• Improve Test Plan (Non UART Features & Combined Tests & change the tests serialization).
8	January 11, 2016	Hany Salah	• Add Buad Rate Testbench calculations. • Modify System Behavioral Description
9	January 19, 2016	Hany Salah	• Add UART specifications (start, stop bits).

Version	Date	Editor	Description
10	January 24, 2016	Hany Salah	• Divide UART fields into commands • Add Environment Configurations
11	January 29, 2016	Hany Salah	• Modify Test Plan • Modify Environment Configurations
12	June 25, 2017	Hany Salah	• Add the coverage plan • Add coverage driven test

2 Design Specifications

2.1 Design Port-list

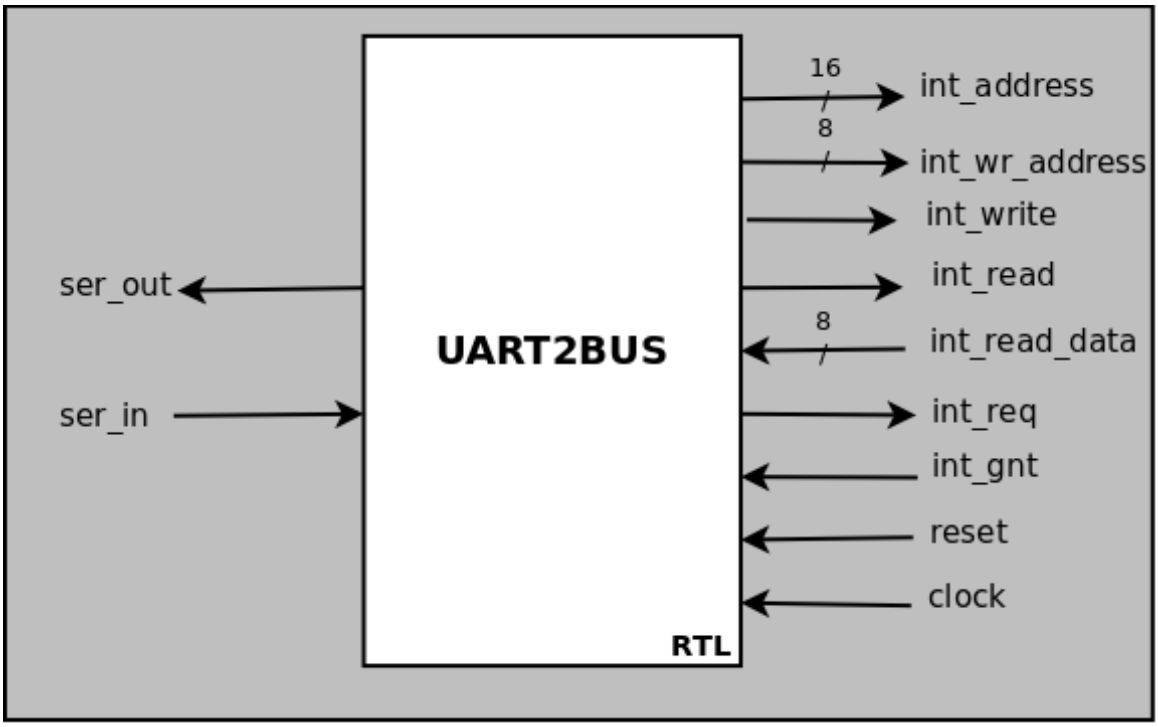

figure 1: Design Entity

Table 2: Design Port-List

Name	Polarity	Width	Direction	Description
clock		1-bit	IN	Global Core Clock signal
reset	high	1-bit	IN	Global Core Asynchronous Reset
int_address		16-bit	OUT	Address Bus To Register File
int_wr_data		8-bit	OUT	Write Data To Register File
int_write	high	1-bit	OUT	Write Control To Register File
int_read	high	1-bit	OUT	Read Control To Register File
int_rd_data		8-bit	IN	Data Read From Register File
int_req	high	1-bit	OUT	Request Internal bus access
int_gnt	high	1-bit	IN	Grant Internal bus access
ser_in		1-bit	IN	Serial Data Input

Name	Polarity	Width	Direction	Description
ser_out		1-bit	OUT	Serial Data Output

2.2 Features

- UART Commands are divided into fields and each field includes one byte or more.
- Each byte is sent in between two standard bits; start and stop.
- Start bit is zero while the stop bit is one.
- Support two modes of operation
  - Text mode command
    - All values are in HEX format.
    - Space is represented as 0x20
    - Tab is represented as 0x09
    - LF is represented as 0x0A
    - CR is represented as 0x0D
    - Both spaces and tabs are considered as white spaces.
    - Both LF and CR are considered as End Of Line (**EOL)**.
    - Includes only two commands; address read and address write commands
      - Address read command
        - 'R' or 'r'.
        - White space single or multiple
        - Address to read in 4-digit HEX
        - CR or LF character
      - On the reception of EOL characters, the core will read the given address and transmit the read value in two HEX character followed by both LF and CR.
      - Address write command
        - 'W' or 'w'.
        - White space single or multiple
        - Data to write in 2-digit HEX
        - White space single or multiple
        - Address to write 4-digit Hex
        - CR or LF character
      - On the reception of EOL characters, the core will write the received data to the given address. No transmission is sent back to the sender.
  - Binary mode command
    - Support single command with configurable number of bytes
    - The first Byte is full of zeros as indicator to the binary command.
    - The second Byte is as following

- Bits [7:6]: not used.
- Bits [5:4]: represent the command type
  - 2'b00[1]: NOP Command, Send ACK if requested.
  - 2'b01: Read Command.
  - 2'b10: Write Command.
  - 2'b11: Invalid.
- Bits [3:2]: not used.
- Bits [1][2]: Auto Increment Enable
  - 1'b0: Auto Increment Enable.
  - 1'b1: Auto Increment Disable.
- Bits [0]: ACK Flag
  - 1'b0: Send no acknowledge byte at the command completion
  - 1'b1: Send acknowledge byte at the command completion

▪ The third byte is used to hold the address high byte.
▪ The forth byte is used to hold the address low byte.
▪ The fifth byte is used to represent the length of buffer to read or write data.
- "0" Value indicates maximum buffer length which is 256 bytes.

▪ The bytes numbered from 6 to (length-1) hold the data and its number should be equal to the fifth byte content[3].

◦ In response to binary command, the responder should react as following.
▪ The byte numbered from 1 to data length which is indicated by the request command is filled with the requested data[4].
▪ The following byte is the acknowledge[5] one and contain the value of 0x5A.

1 NOP Command would be used to verify that the core responds to UART.
2 Auto Increment is often used when reading a buffer from RAM. Otherwise, it's more convenient to turn auto increment off.
3 This field exists only in write commands.
4 This byte exists only is respond to read commands.
5 This byte exists only in case of acknowledge request in the command.

3 System Behavioral Description

UART2BUS module is designed to act as either transmitter or receiver. The main clock that the core uses could be calculated from the following relationship

$$BaudFrequency = \frac{16 * BaudRate}{gcd(GlobalClockFrequecy, 16 * BaudRate)}$$

Where the buad frequency is the actual clock frequency that synchronize both the transmitter and receiver through any data transfer.

Both ser_out & ser_in are high in idel state. Each line is only driven by one driver. When its driver is going to start communication, it pulls it down for one baud clock cycle. It's defined as start bit. After that the sent data will be forced bit by bit[1]. And finally one or two stop bits are followed the last bit. The actual number of stop bits is configured through the VIP configurations, and they all are 1's. UART2BUS polling the ser_in port to capture either the ASCII of 'r', 'R', 'w' and 'W' or the full zero byte otherwise DUT make no response.

In case that the first captured field includes the ASCII of either 'R' or 'r', the DUT expects the next field includes either single or multiple white spaces and then capture the following two fields as the command address. Core wait the following character to be one of EOL and responds with the read data in two hex characters followed by CR and LF characters. Each byte of the sent data should be packed between start and stop bit(s) Any miss-order or non-expected input, core should make no response.

In case that the first field includes the ASCII of either 'W' or 'w', the DUT expects the next field includes either single or multiple white spaces and then capture the following field as the command data. Then it also expects the following field to be single or multiple spaces to capture the next word to be the command address. Finally, core wait the following character to be one of EOL. Any miss-order or non-expected input, core should make no response.

In case that the first field includes zero byte, the DUT capture the following byte. Through this byte, the core detect the command type. In case of invalid command, the core should make no response even if the acknowledge request is activated. The DUT capture the following byte as an address high byte and capture the next byte as an address low byte. The next byte determine the actual data length in bytes. In case of write command, the data bytes follow the length byte. But in case of read command, the command is ended at the end of length byte. Then the core should respond with serial bytes of data equal to the length field in the command. Also according to the acknowledge request bit in the command, the core should follow the data bytes with the unified acknowledge byte.

Regardless of the received command mode or command type, The core will assert int_req and then wait till int_gnt signal assertion and then release int_req. If the received command is write, the core will load int_address port with the command address and assert int_write signal and load int_wr_data with the data corresponding to the loaded address and then deactivate int_write signal to terminate bus usage. If the data includes more than one byte, the core would load int_wr_data

1 Starting with either the MSB or the LSB is configured through the VIP configurations.

port with successive bytes every clock cycle concurrently with increment the port address content and keep int_write signal activated till finalize the data transmission. If the received command is read, the core will load int_address port with the command address and assert int_read signal. The core should wait the requested data on the next clock cycle and then disable int_read to terminate the bus usage. If the requested data includes more than one byte, the incremental address will be loaded to the address port every clock cycle and the corresponding data will be captured through the following clock cycle and so on.

4 Verification Levels & Required Tools

4.1 Verification Levels

- The verification strategy would be black box strategy

4.2 Required Tools

- Sublime Text editor or any other text editor.
- Questasim *Advanced Verification CAD*
- Universal Verification Methodology package

4.3 Environment Configurations

Table 3: Environment Configurations

Field	Description	Possible Choices
Active Edge	The active clock edge at which, the data is changed on the UART buses	• Positive Edge • Negative Edge
First Bit	Represent the sequence through which the byte is serialized	• Most Significant bit • Least Significant bit
Data Mode	The data representation through the text commands	• ASCII • Binary
Number of stop bits	The number of stop bits sent after the latest bit of each byte	• One bit • Two bits
Number of bits	The number of bits through each field transfer	• Seven bits • Eight bits
Parity Mode	The used parity type through each byte.	• Parity Off • Parity Even • Parity Odd

Field	Description	Possible Choices
Response Time	Represent the maximum allowable time through which DUT should respond to the driven request.	No limitations
False Data Enable	Enable force false data on the output port within the read command through the both modes; text, binary	• Yes • No

5 Methodology & Test Scenario

5.1 Methodology

Test-Bench Architecture

We proposed the following test-bench architecture to verify the functionality of UART2BUS module.

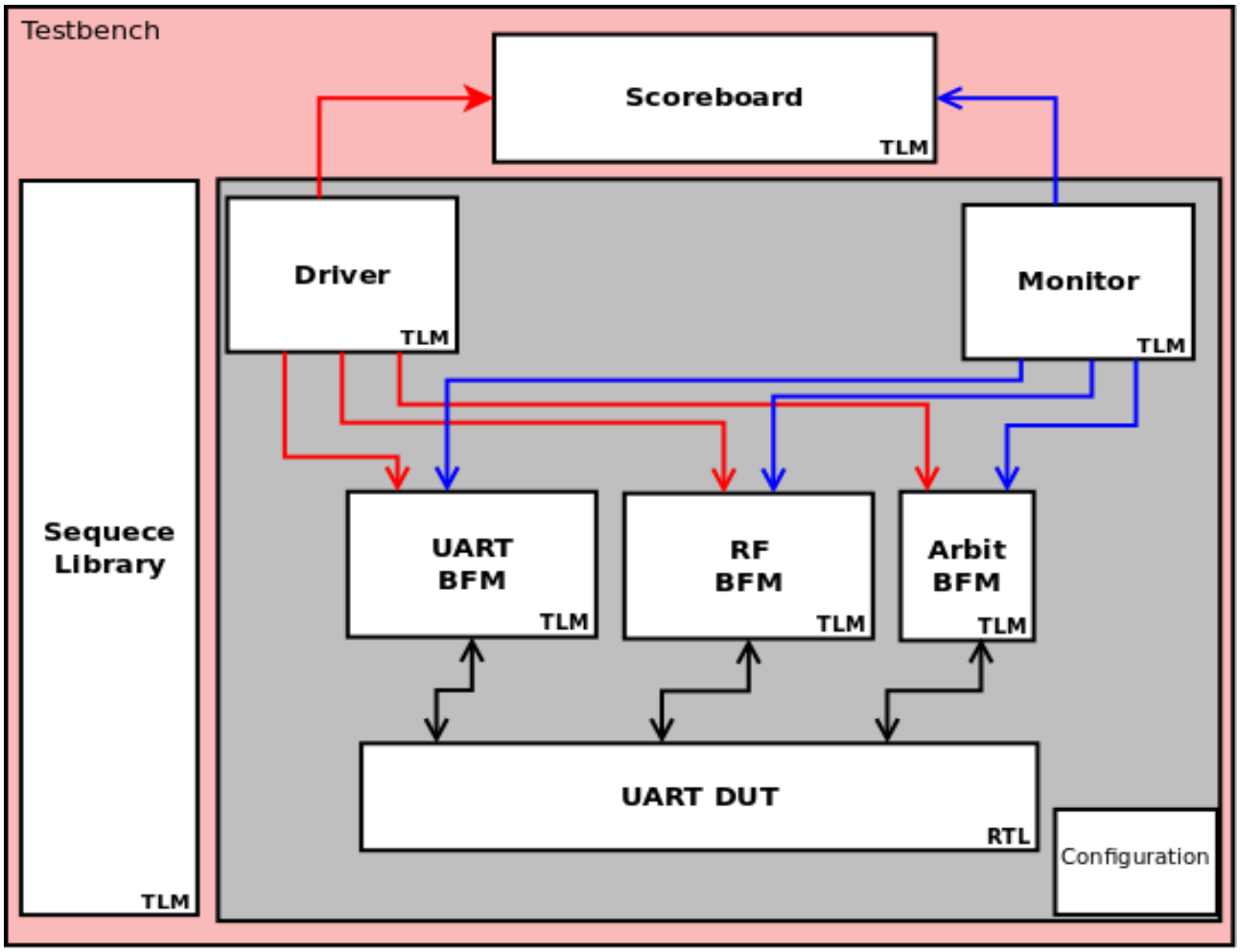

figure 2: TestBench Architecture

- We provide C++ code besides the test-bench to calculate baud frequency to facilitate the usage of UART test-bench.
- Our test-bench will operate on 40 MHz as global clock and 115.2 KHz as baud rate. And then the D-baud frequency is 144 Hz.
- Scoreboard receives transaction from driver and then check the tested address in the register file.
- Bus Functional Model BFM includes three different blocks that are described below

- UART BFM: That act as another UART device and talk with DUT through the two protocol signals.
- RF BFM: That act as memory of 64K bytes and talk with DUT through non-protocol signals
- Arbiter BFM: That act as internal bus arbiter.

Transaction Attributes

We propose the following transaction attributes so that we could implement our test plan

Table 4: Transaction Attributes

Type	Identifier	Description
Enumerate	mode	Represents the mode of command to be one of the following options {text, command, wrong}. Wrong command mode is used to send a tricky fault command to test our DUT.
Byte	prefix_wrong	Represents the wrong prefix that is used in case of wrong mode selection.
Enumerate	space_type1	Represents the type of the used white space in the first field to be one of the following options {single, tab, wrong}. Wrong type also is used to push tricky byte in the text mode.
Enumerate	space_type2	Represents the type of the used white space in the second field to be one of the following options {single, tab, wrong}. Wrong type also is used to push tricky byte in the text mode.
Byte	space_wrong1	Could hold all the 8-bit values except {single or tab} and is used in case of space_type is wrong as the first wrong white space.
Byte	space_wrong2	Could hold all the 8-bit values except {single or tab} and is used in case of space_type is wrong as the second wrong white space.
Dynamic Array of Bytes	_data	Could hold all possible values and its length is constrained to be equal to length_data.
Dynamic Array of Bytes	false_data	Used in case of driving inconsiderable data into the serial output port while the DUT reply the read request command. It could hold all possible values and its length is constrained to be equal to length_data>
Integer Unsigned	length_data	Only in range of [1:256] and equal 1 in case of text command mode.
Enumerate	false_data_en	Enable forcing false data within the read command.

Type	Identifier	Description
Enumerate	eol_type	Represents the type of end of line used to be one of the following choices{cr, lf, wrong}. Wrong type is also used to push DUT in tricky manner.
Byte	eol_wrong	Could hold all 8-bit values except {cr or lf} and is used in case of eol_type is wrong.
Word	address	Could hold all the possible values.
Enumerate	command	Represents the command either to be one of the following choices {read, write, NOP}
Enumerate	ack_req	Represents the acknowledge request{yes, no}
Enumerate	inc_req	Represents the incremental address feature request{yes, no}
Enumerate	char_type	Represents the type of prefix in text mode either to be {capital, small}.
Enumerate	arbit	Represents the internal bus state either {accept, refuse}
Time	Time before	Idle Time before the main command start.
Time	Time after	Idle Time After the finish byte of the main command.

5.2 Testing Scenario

Simple Tests

Simple tests include single command per test in one mode which aim to make sure that the design perform the basic operations correctly.

Table 5: UART Tests

S.N	Feature	Test Procedures
	Write in Text Mode	
1	Could write only a single byte. • 'W' or 'w' • Single space or Tab. • Data • Single space or Tab. • Address • EOL character	1. Apply UART write request using capital W. 2. Apply UART write request using small w. 3. Apply UART write request using single space only. 4. Apply UART write request using tab only. 5. Apply UART write request using both single space and tab. 6. Apply UART write request using one wrong spaces. 7. Apply UART write request using two wrong spaces. 8. Apply UART write request to address 0. 9. Apply UART write request to full range address. 10. Apply UART write request with data equal 0. 11. Apply UART write request with full range data. 12. Apply UART write request using different EOL character. 13. Apply UART Write request using wrong prefix
	Read in Text Mode	
2	Could read only a single byte • 'R' or 'r'. • Single space or tab. • Address • EOL And the response should be • the read data immediately • both LF & CR	1. Apply UART read request using capital R. 2. Apply UART read request using small r. 3. Apply UART read request using single space. 4. Apply UART read request using tab. 5. Apply UART read request using both space and tab. 6. Apply UART read request using one wrong space. 7. Apply UART read request using two wrong spaces 8. Apply UART read request to address 0. 9. Apply UART read request to full range address. 10. Apply UART read request with data equal 0. 11. Apply UART read request with full range data. 12. Apply UART read request using different EOL

S.N	Feature	Test Procedures
		character. 13. Apply UART read request using wrong prefixe
	NOP in Command Mode	
3	Could send NOP command with acknowledge request. • Unified Prefix • Answer should be only a unified acknowledge character. • The address content shouldn't be affected.	1. Apply UART NOP command with acknowledge bit request and right command mode prefix. 2. Apply UART NOP command with wrong command prefix and acknowledge request. 3. Apply several UART NOP command to different locations with different data lengths.
4	Could send NOP command with non acknowledge request acknowledge request • Unified Prefix. • No answer should be exist. • The address content shouldn't be affected regardless of command data.	1. Apply UART NOP command with non-acknowledge bit request and right command mode prefix. 2. Apply UART NOP command with wrong command prefix and non-acknowledge request. 3. Apply several UART NOP command to different locations with different data lengths and non-acknowledge request.
	Write in Command Mode	
5	Could send write command including • Unified Prefix • 16-bit address. • Data length varies from 1 to 256 bytes. • Incremental address ability. • Acknowledge request.	1. Apply UART write command with wrong prefix. 2. Apply UART write commands to different addresses. 3. Apply UART write commands with several data lengths 4. Apply UART write command to address 0 with random data. 5. Apply UART write command to address 0xFFFF with random data. 6. Apply UART write command with acknowledge request. 7. Apply UART write command with non-acknowledge request. 8. Apply UART write command including single byte. 9. Apply UART write command including non-incremental address bit. 10. Apply UART write command including

S.N	Feature	Test Procedures
		incremental address bit.
	Read in Command Mode	
6	Could send read command including • Unified Prefix. • 16-bit addresses. • The length of requested data would vary from 1 to 256 bytes. • Acknowledge request.	1. Apply UART read command with wrong prefix. 2. Apply UART read commands to different addresses. 3. Apply UART read commands with several data lengths 4. Apply UART read command to address 0 with random data. 5. Apply UART read command to address 0xFFFF with random data. 6. Apply UART read command with acknowledge request. 7. Apply UART read command with non-acknowledge request. 8. Apply UART read command including single byte. 9. Apply UART read command including non-incremental address bit. 10. Apply UART read command including incremental address bit.
	Internal Bus	
7	Should request to access the internal bus and wait for grant	1. Apply UART read or write commands and give the UART the bus grant. 2. Apply UART read or write commands and give no agreement to access internal bus
	Invalid Commands[1]	
8	Make no response towards invalid commands	1. Apply Invalid UART command in form of write binary command. 2. Apply Invalid UART command in form of read binary command.

Combined Tests

Combined tests include more than one command per test and aim to put the DUT into highly complicated processes. And also verify the bus functional model besides the DUT.

1 Invalid commands in frame similar to binary command frame

S.N	Feature	Test Procedures
9	Able To receive commands in text mode and command mode without reconfiguration	1. Apply read command in text mode. 2. Apply write command in command mode. 3. Apply read command in command mode. 4. Apply write command in text mode.
10	Able to distinguish between the valid and invalid commands and also able to miss the invalid commands in series of valid commands from different types and modes.	1. Apply four valid commands 2. Apply invalid command 3. Apply valid command. 4. Apply two invalid commands 5. Apply valid command

5.3 Coverage Driven Test

The coverage driven test named “cover_driven_test” is used to randomize the all valid tests (without any wrong white-spaces, EOL, or prefix) and make iterations till the coverage hit the maximum. The minimum coverage requirements are stated in the following table

Table 6: Coverage Threshold in coverage driven test

Covergroup	parameter name	Minimum Threshold
trans_attrib	hit_mode_cov	90%
text_mode_cov	hit_text_cov	90%
binary_mode_cov	hit_bin_cov	90%

To know about the testbench covergroups, refer to the Coverage Statistics

The coverage is being updated using uvm_resources_db mechanism to update the fields “text_coverage”, “binary_coverage”, and “general_coverage” in the database named “coverage_cloud”.

After every iteration, the test layer read the cloud data and build the decision based on the parameters in the table above.

6 Coverage

Table 7: Coverage Analysis

S.N	Covergroup	Coverpoint	Description	Coverbins	Notes
1	trans_attrib	communication_mode	The command mode	text, binary	-
				wrong_mode_text, wrong_mode_bin	illegal_bins[1]
		address	The address of the command	-	-
2	text_mode_cov	command	The type of command in text mode	read, write	
				nop, invalid_read, invalid_write	illegal_bins
		first_white_space_field	The type of the first white space field in text command	Tab, Single Space	
				wrong space	illegal_bins
		second_white_space_field	The type of the first white space field in text command	Tab, Single Space	
				wrong space	illegal_bins
		end_of_line_field	The type of eol character	cr, lf	
				wrong eol	illegal_bins
		prefix_character_type	The type of prefix character R for read W for write	cap,small	
3	binary_mode_cov	command	The type of the command in binary mode	read, write, nop	

1 The DUT is not implemented to overcome invalid wrong inputs like EOL/Whitespace characters and also command prefix. So it is hang-up in such a case. so the invalid input options are put as illegal_bins in the coverage below

S.N	Covergroup	Coverpoint	Description	Coverbins	Notes
				invalid_read, invalid_write	illegal_bins
		acknowledge_require-ment	The ask for acknowledge	yes, no	
		incremental_address_req-uirement	The ask for incremental address	yes, no	
		Length_of_data	The length of binary data within command	zero	
				[1:256]	
				[257:$	illegal_bins