**UNIVERSITY OF MORATUWA**

**Department of Electronic and Telecommunication Engineering Faculty of Engineering**

EN4020 - Advanced Digital Systems

System Bus Design

**Group 01**

|  |  |
| --- | --- |
| 190622R | Tharindu O.K.D |
| 190338C | Kuruppu K.D.S.D. |
| 190072E | Bandara E.M.D.A. |
| 190018V | Abeywickrama K.C.S. |

Date of Submission: 18/12/2023

Contents

[1. Overview 4](#_Toc153752479)

[2. Address Allocation 5](#_Toc153752480)

[3. Master 6](#_Toc153752481)

[3.1. Block Diagram 6](#_Toc153752482)

[3.2. State Diagram 8](#_Toc153752483)

[4. Slave 9](#_Toc153752484)

[4.1. Block Diagram 9](#_Toc153752485)

[4.2. State Diagram 10](#_Toc153752486)

[5. Arbiter 11](#_Toc153752487)

[5.1. Block Diagram 11](#_Toc153752488)

[5.2. State Diagram 15](#_Toc153752489)

[6. Bus Bridge 15](#_Toc153752490)

[6.1. Slave Bus Bridge 16](#_Toc153752491)

[6.2. Master Bus Bridge 17](#_Toc153752492)

[6.3. Asynchronous RX TX 18](#_Toc153752493)

[6.3.1. Transmitter 18](#_Toc153752494)

[6.3.2. Receiver 19](#_Toc153752495)

[7. Timing Analysis Report 20](#_Toc153752496)

[7.1. Constrains 20](#_Toc153752497)

[7.2. Constrains File 21](#_Toc153752498)

[7.3. Clocks Summary 23](#_Toc153752499)

[7.4. Summary (Setup) 23](#_Toc153752500)

[7.5. Summary (Hold) 23](#_Toc153752501)

[7.6. Summary (Recovery) 23](#_Toc153752502)

[7.7. Summary (Removal) 23](#_Toc153752503)

[7.8. Summary (Minimum Pulse Width) 24](#_Toc153752504)

[7.9. Advanced I/O Timing 24](#_Toc153752505)

[7.9.1. Input Transition Times 24](#_Toc153752506)

[7.9.2. Signal Integrity Metrics 25](#_Toc153752507)

[8. Simulation results 26](#_Toc153752508)

[8.1. Master read transaction. 26](#_Toc153752509)

[8.2. Master write transaction. 26](#_Toc153752510)

[8.3. Master goes split. 26](#_Toc153752511)

[8.4. Slave read transaction. 27](#_Toc153752512)

[8.5. Slave write transaction. 27](#_Toc153752513)

[8.6. Slave split transaction. 27](#_Toc153752514)

[8.7. Arbiter master 1 request transaction. 28](#_Toc153752515)

[8.8. Arbiter both master request transaction. 28](#_Toc153752516)

[9. Appendix 29](#_Toc153752517)

[9.1. Master 29](#_Toc153752518)

[9.2. Slave 30](#_Toc153752519)

[9.3. Arbiter 32](#_Toc153752520)

[9.4. Master Bus Bridge 38](#_Toc153752521)

[9.5. Slave Bus Bridge 40](#_Toc153752522)

[9.6. FIFO 41](#_Toc153752523)

# Overview

A diagram of a system

Description automatically generated with medium confidence

Our bus consists of 2 masters, 3 slaves, and 1 bus bridge. Master 1 one has higher priority than master 2. The slaves have different internal memories, and they are denoted below. Furthermore, slave 2 supports split transaction.

|  |  |
| --- | --- |
| **Slave 1** | 2 KB |
| **Slave 2** | 4 KB |
| **Slave 3** | 4 KB |

The slave bus bridge interfaces with the other team’s bus and the masters on our bus views the slave bus bridge as a 10 KB memory space.

The 10 KB memory space for the bus bridge is made up in the following way.

|  |  |
| --- | --- |
| The other bus’s Slave 1 | 2 KB |
| The other bus’s Slave 2 | 4 KB |
| The other bus’s Slave 3 | 4 KB |
| **Total** | 10 KB |

When our team’s bus gets connected with the other team’s bus, one of the masters is replaced with the master bus bridge.

# Address Allocation

|  |  |
| --- | --- |
| **Slave** | **Address** |
| Slave 1 (2K) | 0000 0XXX XXXX XXXX |
| Slave 2 (4K Split Enable) | 0001 XXXX XXXX XXXX |
| Slave 3 (4K) | 0010 XXXX XXXX XXXX |
| Bus Bridge (10K) | 11XX XXXX XXXX XXXX |

A white rectangular object with black text

Description automatically generated

# Master

## A green rectangular object with text Description automatically generatedBlock Diagram

|  |  |
| --- | --- |
| **Input Pin** | **Description** |
| ack | Acknowledgment coming from arbiter to indicate that sent address is valid. |
| bgrant | Indicate that master has access to bus. |
| clk | Clock signal. |
| m\_addr[15..0] | Address given to master from FPGA switches. |
| m\_mode | Select mode of transaction. Read = 0, Write =1. Given from push buttons. |
| m\_start | Start the transaction. Connected to FPGA push button. |
| m\_wr\_data[7..0] | Data sending through bus in write transactions. Connected to data\_out port of BRAM. |
| rd\_bus | Serial data input bus port. |
| rstn | Asynchronous active low reset signal. |
| slave\_ready | Indicate slave is ready to receive. |
| slave\_valid | Indicate data on rd\_bus is valid. |
| split | Indicate slave goes to split mode. |
| **Output pin** | **Description** |
| breq | Request the bus. |
| m\_rd\_data[7..0] | Data received from read transactions. Connected to BRAM data\_in port. |
| m\_wr\_en | Write enable signal for BRAM to store the data. |
| master\_ready | Indicate master is ready to receive. |
| master\_valid | Indicate data in wr\_bus is valid. |
| mode | Read (mode = 0), write (mode = 1) |
| wr\_bus | Serial data output bus. |

## State Diagram

# Slave

## Block Diagram

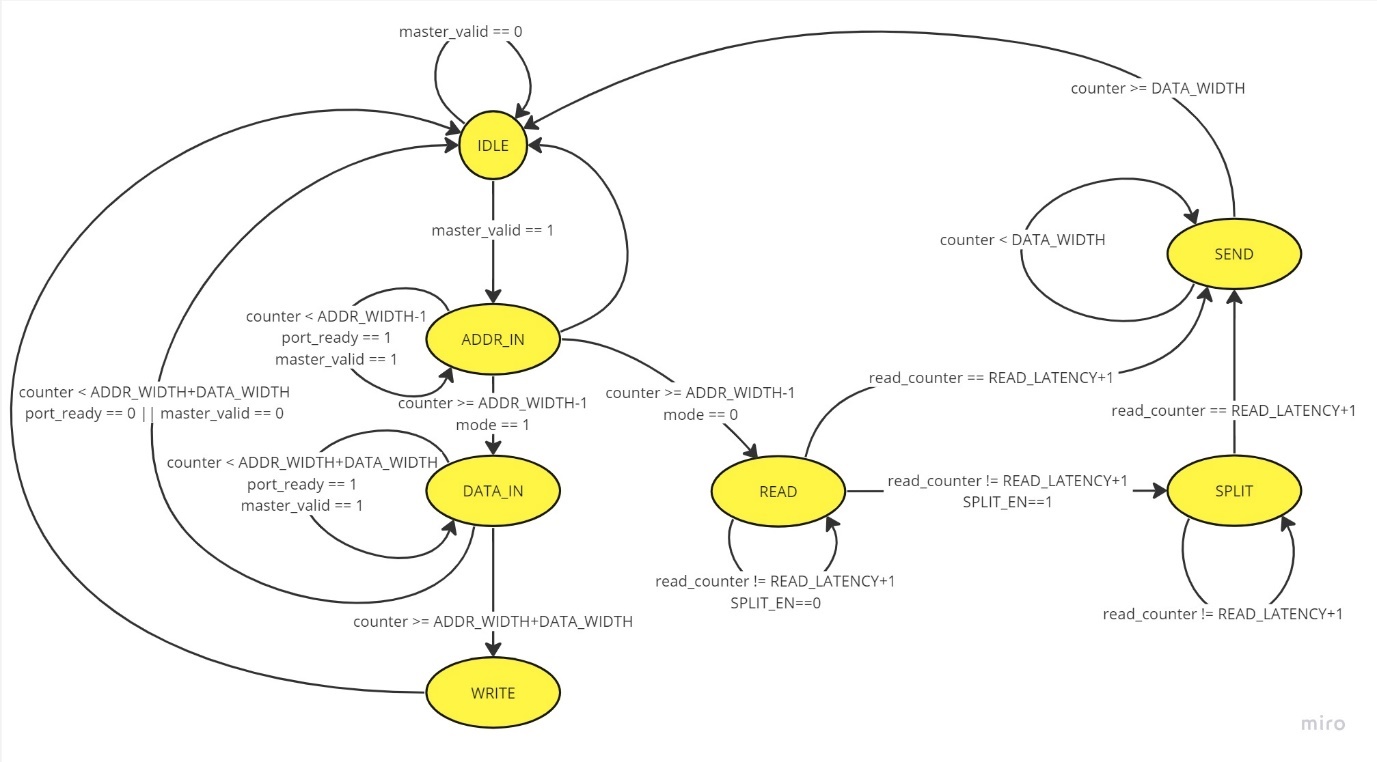
A screenshot of a computer program

Description automatically generated

|  |  |
| --- | --- |
| **Input Pin** | **Description** |
| clk | Clock signal |
| master\_ready | Indicates master is ready to receive data |
| master\_valid | Indicates the data in the wr\_bus is valid |
| mode | Read (mode = 0), write (mode = 1) |
| ram\_in[7..0] | Register to load the contents from slave BRAM memory |
| rstn | Active low reset signal |
| wr\_bus | Serial data in port |

|  |  |
| --- | --- |
| **Output Pin** | **Description** |
| ram\_addr\_out[11..0] | Register to address the BRAM memory |
| ram\_out[7..0] | Register to send data to the BRAM memory |
| ram\_wr\_en | Register to enable writing to the BRAM memory |
| rd\_bus | Serial data out port |
| slave\_ready | Indicates the slave is ready to receive data |
| slave\_valid | Indicates the data in the rd\_bus is valid |
| split | Indicates to the arbiter to split the transaction |

## State Diagram



# Arbiter

## Block Diagram

A green line with white text

Description automatically generated

|  |  |
| --- | --- |
| **Input Pin** | **Description** |
| bb\_rd\_bus | Bus bridge serial data out bus. |
| bb\_slave\_ready | Indicate bus bridge is ready to receive. |
| bb\_salve\_valid | Indicate data on bb\_rd\_bus is valid. |
| clk | Clock signal |
| m1\_breq | Request signal of Master 1. |
| m1\_master\_ready | Master 1 is ready to receive. |
| m1\_master\_valid | Data on m1\_wr\_bus is valid. |
| m1\_mode | Read (mode = 0), write (mode = 1) |
| m1\_wr\_bus | Master 1 serial data out bus. |
| m2\_breq | Request signal of Master 2. |
| m2\_master\_ready | Master 2 is ready to receive. |
| m2\_master\_valid | Data on m2\_wr\_bus is valid. |
| m2\_mode | Read (mode = 0), write (mode = 1) |
| m2\_wr\_bus | Master 2 serial data out bus. |
| rstn | Asynchronous active low reset signal. |
| slave\_split | Indicate slave 2 is going to split mode. |
| s1\_rd\_bus | Slave 1 serial data out port. |
| s1\_slave\_ready | Slave 1 is ready to receive. |
| s1\_slave\_valid | Data on s1\_rd\_bus is valid. |
| s2\_rd\_bus | Slave 2 serial data out port. |
| s2\_slave\_ready | Slave 2 is ready to receive. |
| s2\_slave\_valid | Data on s2\_rd\_bus is valid. |
| s3\_rd\_bus | Slave 3 serial data out port. |
| s3\_slave\_ready | Slave 3 is ready to receive. |
| s3\_slave\_valid | Data on s3\_rd\_bus is valid. |
| **Output Pin** | **Description** |
| bb\_master\_ready | Master is ready to receive data from bus bridge. |
| bb\_master\_valid | Data on bb\_wr\_bus is valid. |
| bb\_mode | Read (mode = 0), write (mode = 1) |
| bb\_wr\_bus | Serial data in bus for bus bridge. |
| m1\_ack | Acknowledgment for Master 1. |
| m1\_bgrant | Bus grant signal for Master 1. |
| m1\_rd\_bus | Master 1 serial data in bus. |
| m1\_slave\_ready | Slave is ready to receive the data from master 1. |
| m1\_slave\_valid | Data on m1\_rd\_bus is valid. |
| m1\_split | Indicate master 1 that slave goes to split. |
| m2\_ack | Acknowledgment for Master 2. |
| m2\_bgrant | Bus grant signal for Master 2. |
| m2\_rd\_bus | Master 2 serial data in bus. |
| m2\_slave\_ready | Slave is ready to receive the data from master 2. |
| m2\_slave\_valid | Data on m2\_rd\_bus is valid. |
| m2\_split | Indicate master 2 that slave goes to split. |
| s1\_master\_ready | Master is ready to receive data from slave 1. |
| s1\_master\_valid | Data on s1\_wr\_bus is valid. |
| s1\_mode | Read (mode = 0), write (mode = 1) |
| s1\_wr\_bus | Serial data in bus for slave 1. |
| s2\_master\_ready | Master is ready to receive data from slave 2. |
| s2\_master\_valid | Data on s2\_wr\_bus is valid. |
| s2\_mode | Read (mode = 0), write (mode = 1) |
| s2\_wr\_bus | Serial data in bus for slave 2. |
| s3\_master\_ready | Master is ready to receive data from slave 3. |
| s3\_master\_valid | Data on s3\_wr\_bus is valid. |
| s3\_mode | Read (mode = 0), write (mode = 1) |
| s3\_wr\_bus | Serial data in bus for slave 3. |

## State Diagram

# Bus Bridge

There are two bus bridge modules:

* Slave bus bridge
* Master bus bridge

The slave bus bridge is a slave to our bus, and it interfaces with a master bus bridge in the other team’s bus via UART. This interface is used to send transaction commands to slaves and receive data from slaves for READ requests in the other team’s bus.

The master bus bridge is a master to our bus, and it interfaces with a slave bus bridge in the other team’s bus via UART. This interface is used to receive transaction commands from masters and send data to masters for READ requests in the other team’s bus.

The exchange of transaction commands is done through a UART port that facilitates transmission/reception of 25-bit width words per single transmission/reception. The 25-bit word is arranged in the following manner.

|  |  |  |
| --- | --- | --- |
| 1 | 16 | 8 |
| Mode | Address | Data |

The exchange of data in a READ transaction is done through a UART port that facilitate transmission/reception of 8-bit width words per single transmission/reception. The entire 8-bit word is the data that need to be transmitted/received.

## A screenshot of a computer program Description automatically generatedSlave Bus Bridge

|  |  |
| --- | --- |
| **Input Pin** | **Description** |
| clk | Clock signal |
| master\_ready | Indicates master is ready to receive data |
| master\_valid | Indicates the data in the wr\_bus is valid |
| mode | Read (mode = 0), write (mode = 1) |
| rstn | Active low reset signal |
| uart\_register\_in[7..0] | Register that holds the incoming data from uart |
| valid\_in | Indicates the data in the uart\_register\_in register is valid |
| wr\_bus | Serial data in port |
| **Output Pin** | **Description** |
| rd\_bus | Serial data out port |
| slave\_ready | Indicates the slave is ready to receive data |
| slave\_valid | Indicates the data in the rd\_bus is valid |
| uart\_register\_out[24..0] | Register to hold the data that need to be sent via uart |
| valid\_out | Indicates that the data in the uart\_register\_out is valid |

## Master Bus Bridge

A screenshot of a computer program

Description automatically generated

|  |  |
| --- | --- |
| **Input Pin** | **Description** |
| ack | Acknowledgment coming from arbiter to indicate that sent address is valid. |
| bgrant | Indicate that master has access to bus. |
| clk | Clock signal |
| fifo\_data\_in[24..0] | Data coming from FIFO. |
| fifo\_empty | Indicate that FIFO is empty. |
| rd\_bus | Serial data input bus port. |
| rstn | Asynchronous active low reset signal |
| slave\_ready | Slave is ready to receive. |
| slave\_valid | Data on rd\_bus is valid. |
| split | Indicate that slave is going to split. |

|  |  |
| --- | --- |
| **Output Pin** | **Description** |
| breq | Request bus access from arbiter. |
| fifo\_deq | Dequeue data from FIFO. |
| m\_out\_valid | Data on uart\_register\_out is valid. |
| master\_ready | Bus bridge is ready to receive. |
| master\_valid | Data on wr\_bus is valid. |
| mode | Read (mode = 0), write (mode = 1) |
| uart\_register\_out[7..0] | Data received from read transaction. |
| wr\_bus | Serial data out bus. |

## Asynchronous RX TX

### Transmitter

A green rectangle with white text

Description automatically generated

|  |  |
| --- | --- |
| **Input Pin** | **Description** |
| clk | Clock signal |
| data\_tx[24..0] | Register to hold the data that need to be transmitted serially |
| rstn | Active low reset signal |
| valid\_tx | Indicates the data in the data\_tx register is valid and that data can be transmitted |

|  |  |
| --- | --- |
| **Output Pin** | **Description** |
| ready\_tx | Indicates whether the module is busy with a transaction.  (Busy:1, Not Busy:0) |
| sig\_tx | Serial data out port |

### Receiver

A green rectangular object with white text

Description automatically generated

|  |  |
| --- | --- |
| **Input Pin** | **Description** |
| clk | Clock signal |
| ready\_rx | Indicates whether the received data has been recorded by another module (Recorded: 1, Not Recorded: 0) |
| rstn | Active low reset signal |
| sig\_rx | Serial data in port |

|  |  |
| --- | --- |
| **Output Pin** | **Description** |
| data\_rx[24..0] | Register to hold the data that is received |
| valid\_rx | Indicates the data in the data\_rx register is valid and can be read by another module |

# Timing Analysis Report

## Constrains

|  |  |
| --- | --- |
| **Constrains** | **Value (ns)** |
| Clock Period | 60 |
| Clock Uncertainty | 3 (5%) |
| Input Delay | 24 (40%) |
| Output Delay | 24 (40%) |

## Constrains File

1. #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

2. # Time Information

3. #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

4.

5. set\_time\_format -unit ns -decimal\_places 3

6.

7. #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

8. # Create Clock

9. #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

10. create\_clock -name {clk} -period 60.000 -waveform { 0.000 30.000 } [get\_ports {clk}]

11.

12. #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

13. # Create Generated Clock

14. #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

15.

16. #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

17. # Set Clock Latency

18. #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

19.

20. #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

21. # Set Clock Uncertainty

22. #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

23.

24. set\_clock\_uncertainty -rise\_from [get\_clocks {clk}] -rise\_to [get\_clocks {clk}] -setup 3.000

25. set\_clock\_uncertainty -rise\_from [get\_clocks {clk}] -fall\_to [get\_clocks {clk}] -setup 3.000

26. set\_clock\_uncertainty -fall\_from [get\_clocks {clk}] -rise\_to [get\_clocks {clk}] -setup 3.000

27. set\_clock\_uncertainty -fall\_from [get\_clocks {clk}] -fall\_to [get\_clocks {clk}] -setup 3.000

28.

29. #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

30. # Set Input Delay

31. #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

32.

33. set\_input\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {addr[0]}]

34. set\_input\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {addr[1]}]

35. set\_input\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {addr[2]}]

36. set\_input\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {addr[3]}]

37. set\_input\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {addr[4]}]

38. set\_input\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {addr[5]}]

39. set\_input\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {addr[6]}]

40. set\_input\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {addr[7]}]

41. set\_input\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {addr[8]}]

42. set\_input\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {addr[9]}]

43. set\_input\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {addr[10]}]

44. set\_input\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {addr[11]}]

45. set\_input\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {addr[12]}]

46. set\_input\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {addr[13]}]

47. set\_input\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {addr[14]}]

48. set\_input\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {addr[15]}]

49. set\_input\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {keysn[0]}]

50. set\_input\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {keysn[1]}]

51. set\_input\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {keysn[2]}]

52. set\_input\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {keysn[3]}]

53. set\_input\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {m\_ready\_rx}]

54. set\_input\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {m\_sig\_rx}]

55. set\_input\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {s\_ready\_rx}]

56. set\_input\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {s\_sig\_rx}]

57.

58. #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

59. # Set Output Delay

60. #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

61.

62. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {hex0[0]}]

63. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {hex0[1]}]

64. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {hex0[2]}]

65. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {hex0[3]}]

66. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {hex0[4]}]

67. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {hex0[5]}]

68. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {hex0[6]}]

69. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {hex1[0]}]

70. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {hex1[1]}]

71. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {hex1[2]}]

72. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {hex1[3]}]

73. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {hex1[4]}]

74. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {hex1[5]}]

75. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {hex1[6]}]

76. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {hex2[0]}]

77. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {hex2[1]}]

78. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {hex2[2]}]

79. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {hex2[3]}]

80. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {hex2[4]}]

81. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {hex2[5]}]

82. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {hex2[6]}]

83. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {hex3[0]}]

84. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {hex3[1]}]

85. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {hex3[2]}]

86. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {hex3[3]}]

87. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {hex3[4]}]

88. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {hex3[5]}]

89. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {hex3[6]}]

90. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {m1\_ack\_led}]

91. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {m1\_master\_ready\_led}]

92. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {m1\_master\_valid\_led}]

93. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {m1\_mode\_led}]

94. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {m1\_slave\_ready\_led}]

95. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {m1\_slave\_valid\_led}]

96. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {m2\_mode\_led}]

97. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {m\_ready\_tx}]

98. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {m\_sig\_tx}]

99. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {rstn\_led}]

100. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {s1\_master\_ready\_led}]

101. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {s1\_master\_valid\_led}]

102. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {s1\_slave\_ready\_led}]

103. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {s1\_slave\_valid\_led}]

104. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {s\_ready\_tx}]

105. set\_output\_delay -add\_delay  -clock [get\_clocks {clk}]  24.000 [get\_ports {s\_sig\_tx}]

106.

107. #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

108. # Set Clock Groups

109. #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

110.

111. #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

112. # Set False Path

113. #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

114.

115. #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

116. # Set Multicycle Path

117. #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

118.

119. #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

120. # Set Maximum Delay

121. #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

122.

123. #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

124. # Set Minimum Delay

125. #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

126.

127. #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

128. # Set Input Transition

129. #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

130.

131.

## A screenshot of a computer Description automatically generatedClocks Summary

## A screenshot of a computer Description automatically generatedSummary (Setup)

## Summary (Hold)A screenshot of a computer Description automatically generated

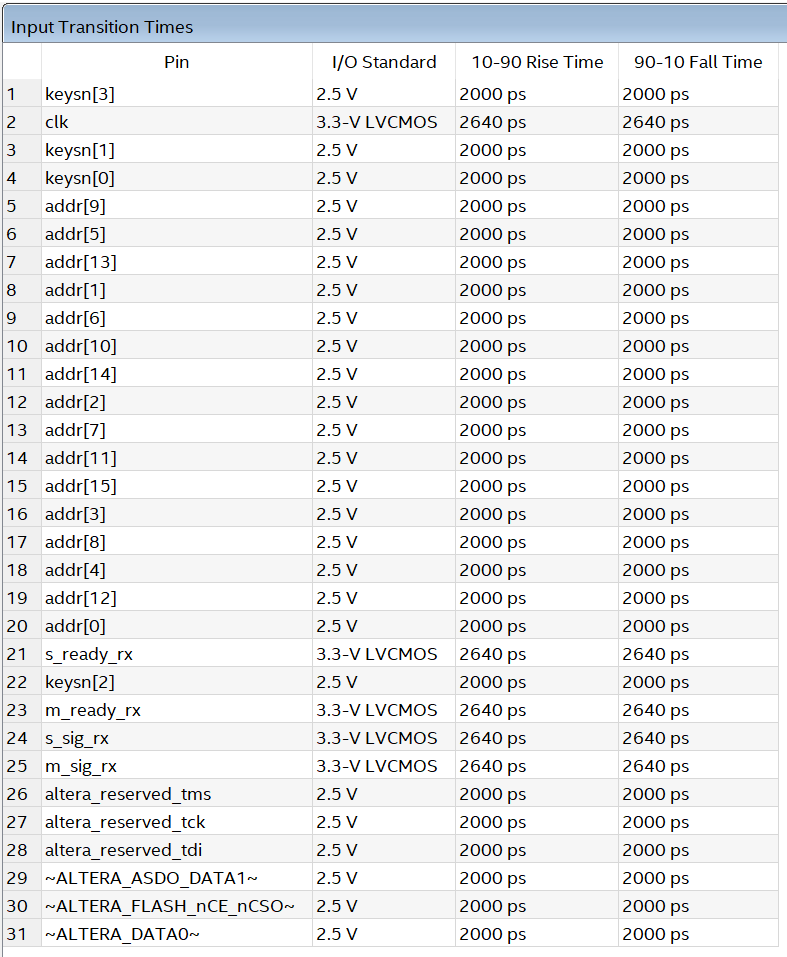
## Summary (Recovery)

## Summary (Removal)

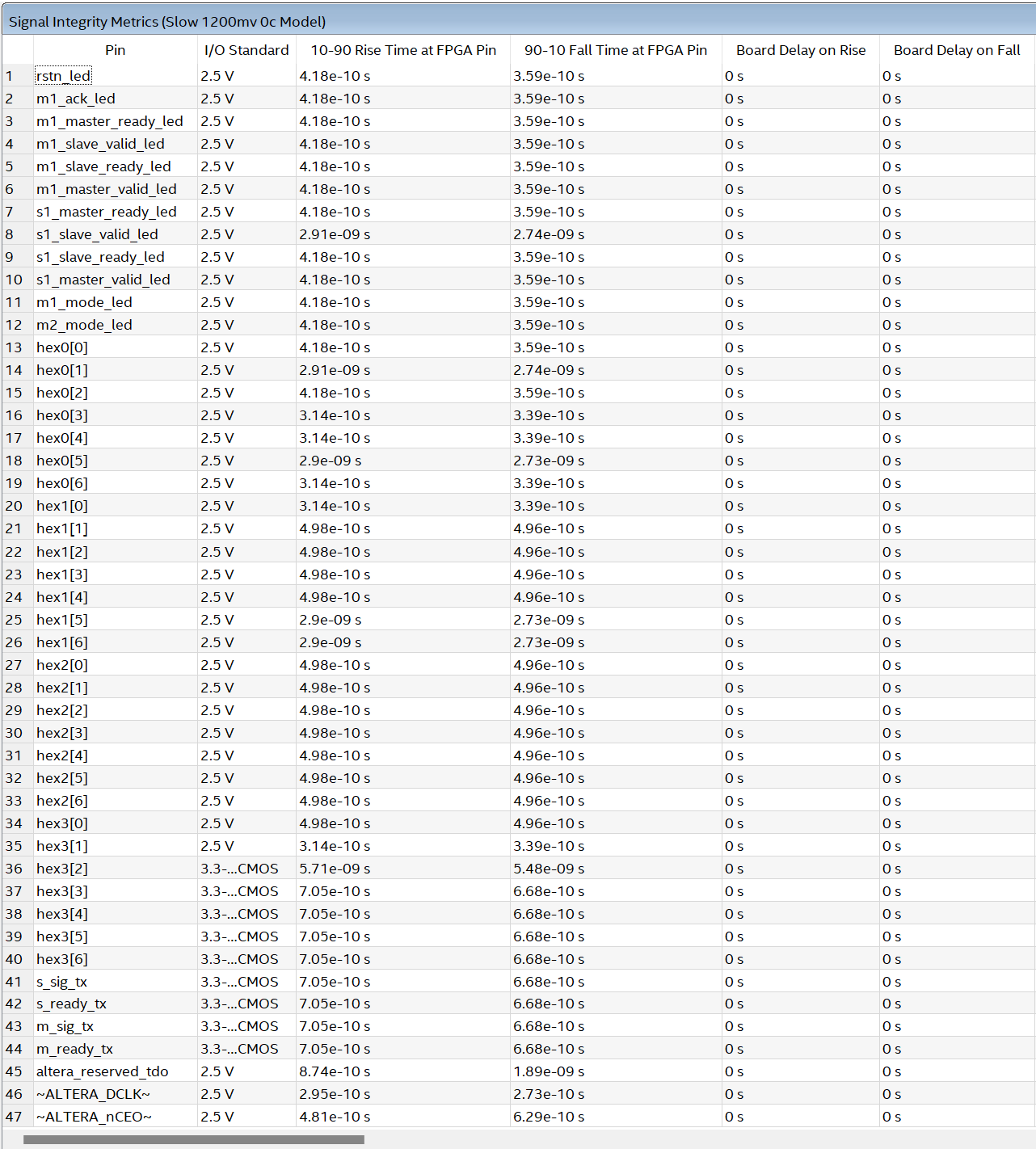
## Summary (Minimum Pulse Width)

## Advanced I/O Timing

### Input Transition Times

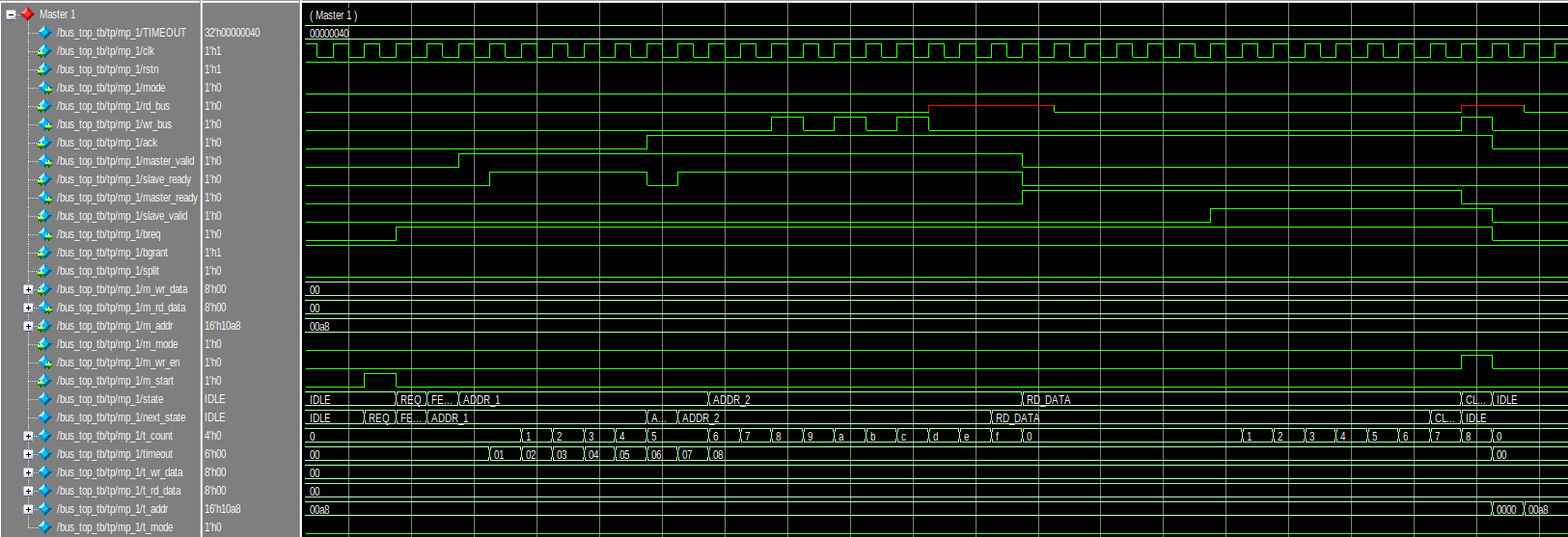


### Signal Integrity Metrics

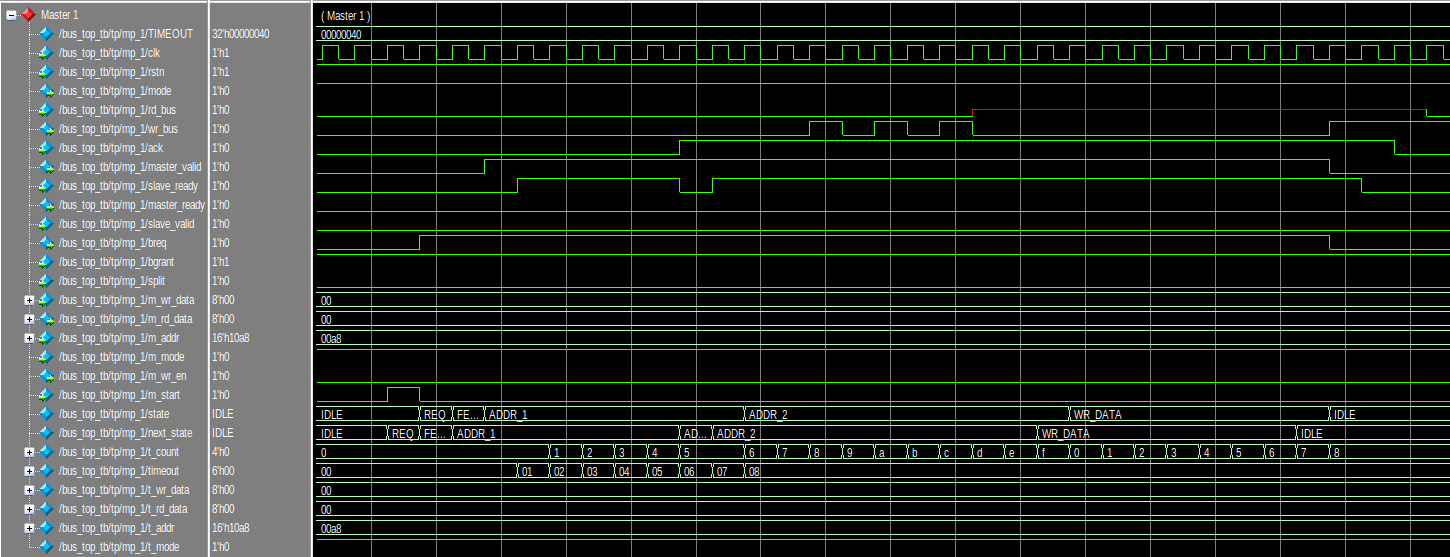


# Simulation results

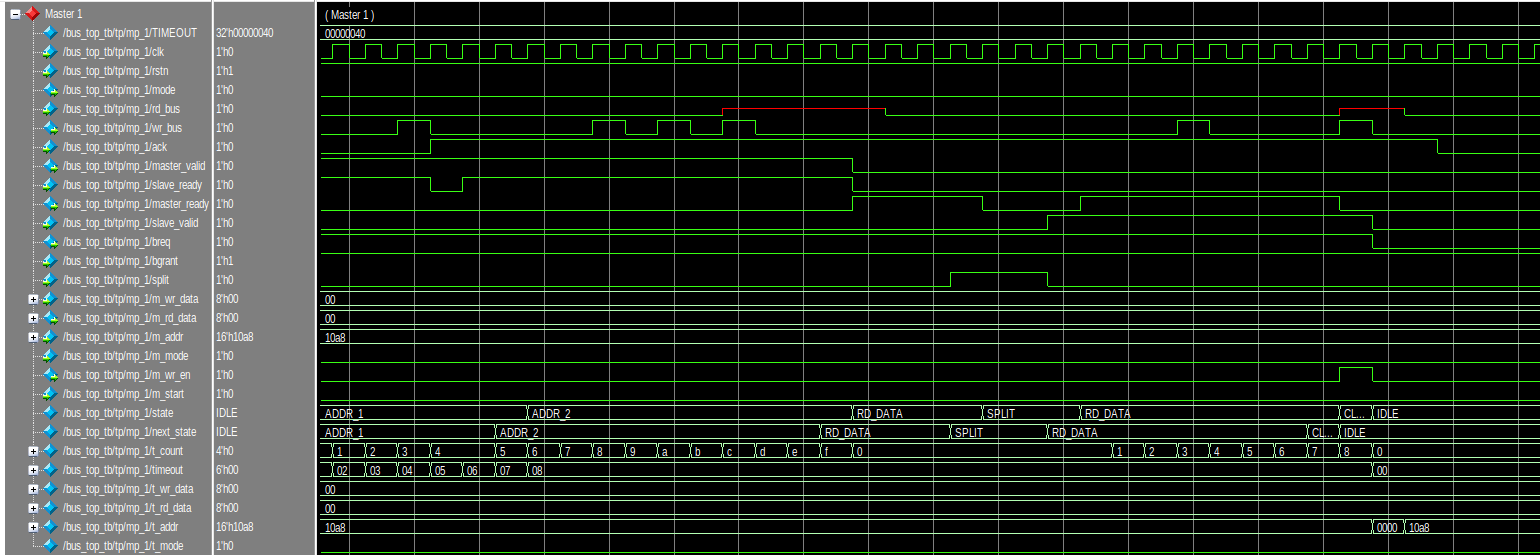
## Master read transaction.



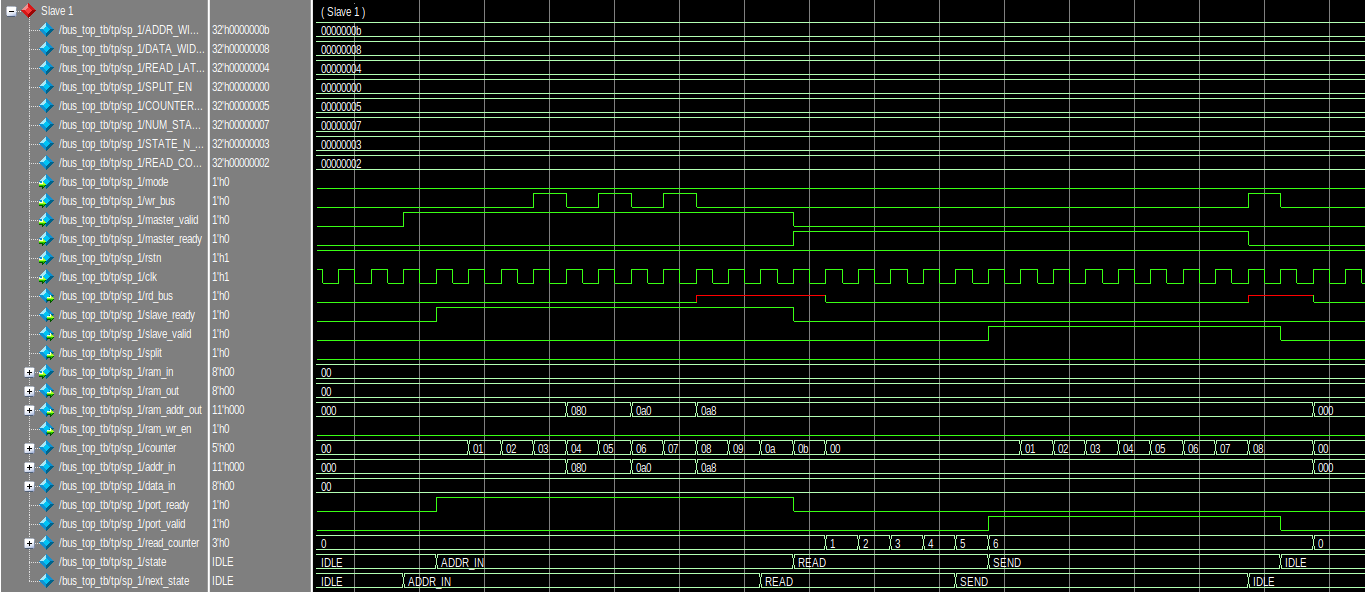
## Master write transaction.



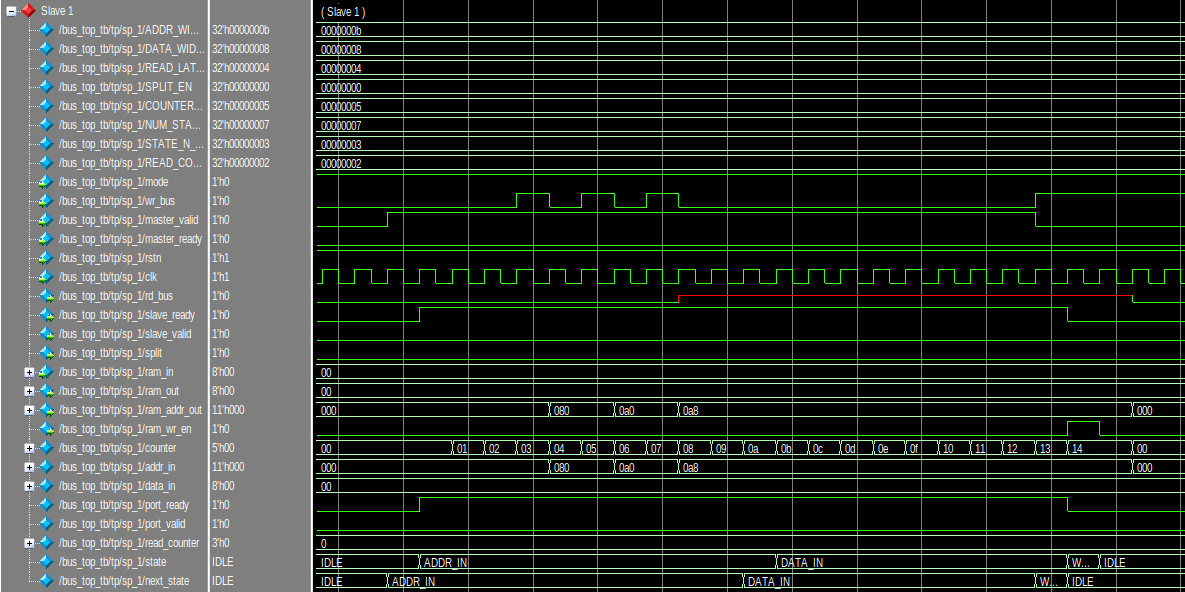
## Master goes split.



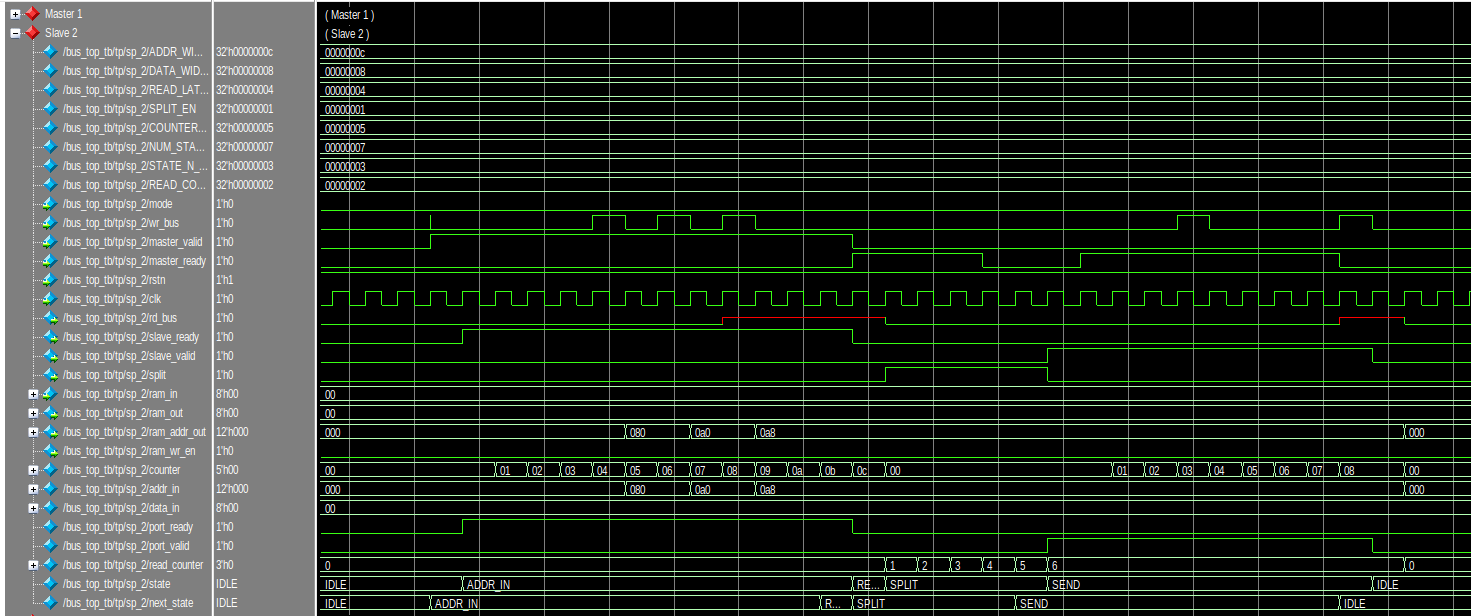
## Slave read transaction.



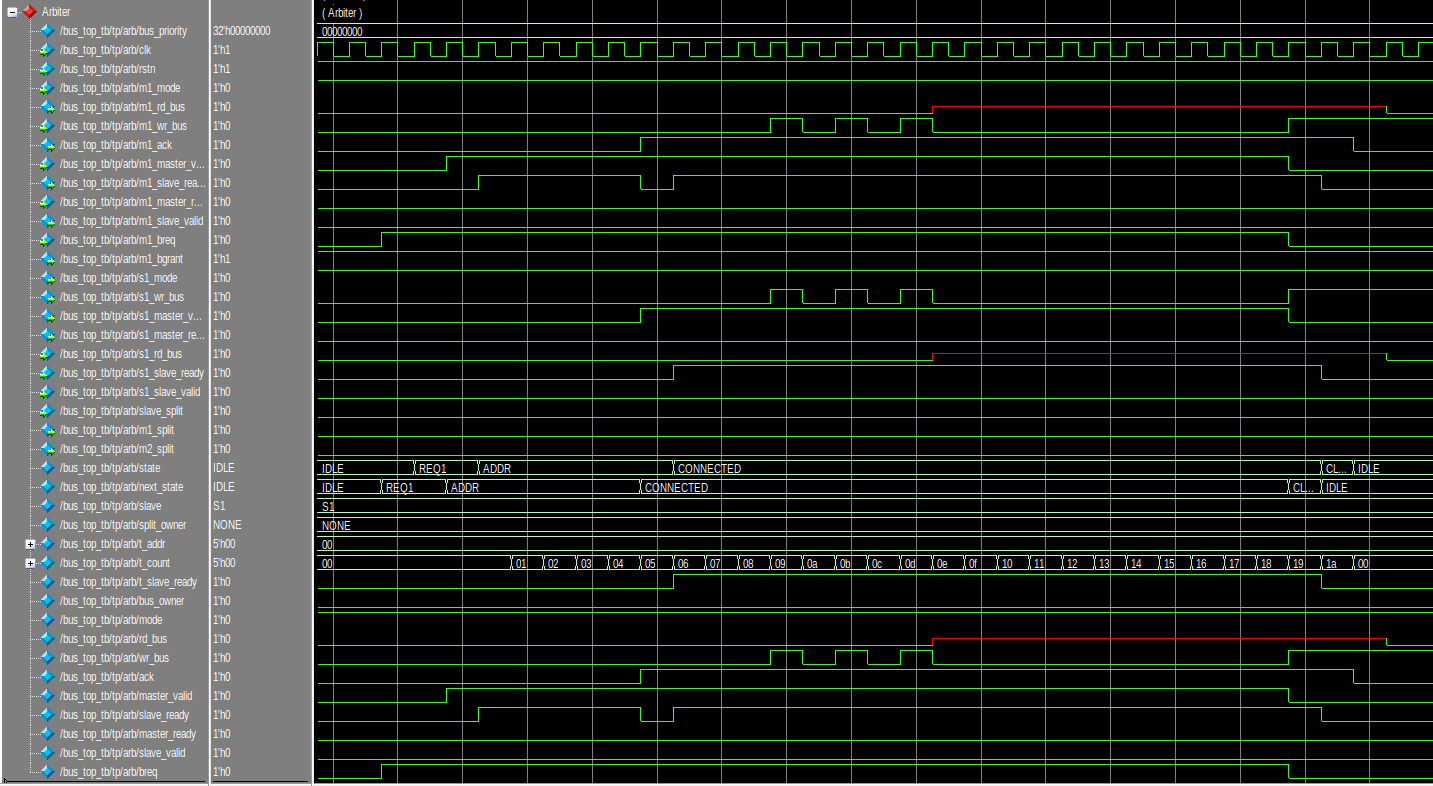
## Slave write transaction.



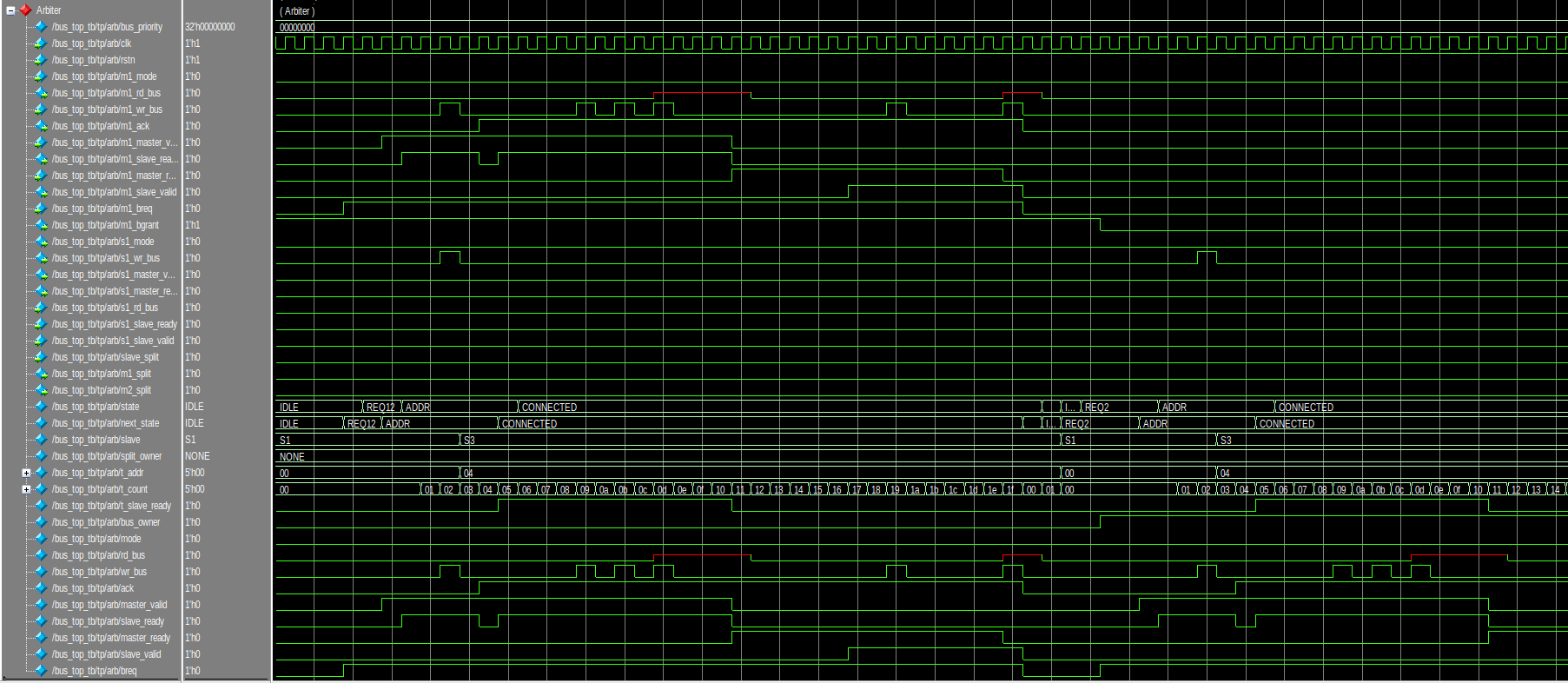
## Slave split transaction.



## Arbiter master 1 request transaction.



## Arbiter both master request transaction.



# Appendix

## Master

1. module master\_port (

2.     input   logic       clk,

3.     input   logic       rstn,

4.     // connections to bus

5.     output  logic       mode,

6.     input   logic       rd\_bus,

7.     output  logic       wr\_bus,

8.     input   logic       ack,

9.     output  logic       master\_valid,

10.     input   logic       slave\_ready,

11.     output  logic       master\_ready,

12.     input   logic       slave\_valid,

13.     output  logic       breq,

14.     input   logic       bgrant,

15.     input   logic       split,

16.

17.     // connections to master

18.     input   logic[7:0]  m\_wr\_data,

19.     output  logic[7:0]  m\_rd\_data,

20.     input   logic[15:0] m\_addr,

21.     input   logic       m\_mode,

22.     output  logic       m\_wr\_en,

23.     input   logic       m\_start

24. );

25.     enum logic[3:0] {IDLE, REQ, FETCH, ADDR\_1, ADDR\_2, WR\_DATA, RD\_DATA, CLEAN, SPLIT, TIMEOUT\_STATE} state, next\_state;

26.     localparam TIMEOUT = 64;

27.

28.     logic[3:0]  t\_count;

29.     logic[$clog2(TIMEOUT)-1:0] timeout;

30.     logic[7:0]  t\_wr\_data;

31.     logic[7:0]  t\_rd\_data;

32.     logic[15:0] t\_addr;

33.     logic       t\_mode;

34.

35.     always\_comb begin : NEXT\_STATE\_LOGIC

36.         case (state)

37.             IDLE:               next\_state = m\_start ? REQ : IDLE;

38.             REQ:                next\_state = bgrant ? FETCH : REQ;

39.             FETCH:              next\_state = ADDR\_1;

40.             ADDR\_1:             next\_state = timeout != TIMEOUT - 1 ? ((t\_count == 5 & slave\_ready) ? (ack ? ADDR\_2 : CLEAN) : ADDR\_1) : TIMEOUT\_STATE;

41.             TIMEOUT\_STATE:      next\_state = REQ;

42.             ADDR\_2:             next\_state = (t\_count == 15 & slave\_ready) ? (t\_mode ? WR\_DATA : RD\_DATA) : ADDR\_2;

43.             WR\_DATA:            next\_state = (t\_count == 7  & slave\_ready) ? IDLE : WR\_DATA;

44.             RD\_DATA:            next\_state = split == 0 ? ((t\_count == 7 & slave\_valid) ? CLEAN : RD\_DATA) : SPLIT;

45.             SPLIT:              next\_state = split ? SPLIT : RD\_DATA;

46.             CLEAN:              next\_state = IDLE;

47.             default:            next\_state = IDLE;

48.         endcase

49.     end

50.

51.     always\_ff @(posedge clk or negedge rstn) begin : STATE\_SEQUENCER

52.         state <= !rstn ? IDLE : next\_state;

53.     end

54.

55.     always\_comb begin : OUTPUT\_LOGIC

56.         wr\_bus  = state == WR\_DATA ? t\_wr\_data[7] : t\_addr[15 - t\_count];

57.         mode    = t\_mode;

58.         master\_valid = state == ADDR\_2 || state == ADDR\_1 || state == WR\_DATA;

59.         master\_ready = state == RD\_DATA;

60.         m\_rd\_data = t\_rd\_data;

61.         m\_wr\_en = state == CLEAN & t\_count == 8;

62.         breq = state != IDLE && state != TIMEOUT\_STATE;

63.     end

64.

65.     always\_ff @(posedge clk or negedge rstn) begin : REG\_LOGIC

66.         if (!rstn) begin

67.             t\_count   <= 0;

68.             t\_wr\_data <= 0;

69.             t\_rd\_data <= 0;

70.             t\_addr    <= 0;

71.             t\_mode    <= 0;

72.             timeout   <= 0;

73.         end else begin

74.             case (state)

75.                 IDLE: begin

76.                     t\_wr\_data <= m\_wr\_data;

77.                     t\_addr    <= m\_addr;

78.                     t\_mode    <= m\_mode;

79.                 end

80.

81.                 REQ: begin

82.                     timeout <= 0;

83.                     t\_count <= 0;

84.                 end

85.

86.                 ADDR\_1:begin

87.                     timeout <= timeout + 1;

88.

89.                     if(slave\_ready) begin

90.                         t\_count <= t\_count + 1;

91.                     end

92.                 end

93.

94.                 ADDR\_2: if(slave\_ready) begin

95.                     t\_count <= t\_count + 1;

96.                 end

97.

98.                 WR\_DATA: if(slave\_ready) begin

99.                     t\_wr\_data <= t\_wr\_data << 1;

100.                     t\_count <= t\_count + 1;

101.                 end

102.

103.                 RD\_DATA: if(slave\_valid) begin

104.                     t\_rd\_data <= {t\_rd\_data[6:0], rd\_bus};

105.                     t\_count <= t\_count + 1;

106.                 end

107.

108.                 CLEAN: begin

109.                     t\_count <= 0;

110.                     t\_wr\_data <= 0;

111.                     t\_rd\_data <= 0;

112.                     t\_addr    <= 0;

113.                     t\_mode    <= 0;

114.                     timeout   <= 0;

115.                 end

116.             endcase

117.         end

118.     end

119. endmodule

## Slave

1. module slave\_port\_v2#(

2.     parameter ADDR\_WIDTH = 16,

3.     parameter DATA\_WIDTH = 8,

4.     parameter READ\_LATENCY = 4,

5.     parameter SPLIT\_EN = 0

6. )(

7.     input logic mode, wr\_bus, master\_valid, master\_ready, rstn, clk,

8.     output logic rd\_bus, slave\_ready, slave\_valid, split,

9.     input   logic[DATA\_WIDTH-1:0]   ram\_in,

10.     output  logic[DATA\_WIDTH-1:0]   ram\_out,

11.     output  logic[ADDR\_WIDTH-1:0]   ram\_addr\_out,

12.     output  logic                   ram\_wr\_en

13. );

14.

15. // local parameters

16. localparam COUNTER\_LENGTH = $clog2(ADDR\_WIDTH+DATA\_WIDTH);

17. localparam NUM\_STATES = 7;

18. localparam STATE\_N\_BITS = $clog2(NUM\_STATES);

19. localparam READ\_COUNTER\_LENGTH = $clog2(READ\_LATENCY);

20.

21. // internal signals

22. logic [COUNTER\_LENGTH-1:0]counter;

23. logic [ADDR\_WIDTH-1:0]addr\_in;

24. logic [DATA\_WIDTH-1:0]data\_in;

25. logic port\_ready, port\_valid;

26. logic [READ\_COUNTER\_LENGTH:0]read\_counter;

27.

28. // definition of states

29. enum logic[STATE\_N\_BITS-1:0] {IDLE, ADDR\_IN, DATA\_IN, WRITE, READ, SEND, SPLIT} state, next\_state;

30.

31. assign slave\_ready = port\_ready;

32. assign slave\_valid = port\_valid;

33. assign ram\_wr\_en = (state == WRITE);

34. assign ram\_addr\_out = addr\_in;

35. assign ram\_out = data\_in;

36.

37. always\_comb begin : NEXT\_STATE\_DECODER

38.     case (state)

39.         IDLE: next\_state = ( ( master\_valid == 1 ) ? ADDR\_IN : IDLE );

40.         ADDR\_IN: next\_state = ( (counter < ADDR\_WIDTH-1) ? ( (port\_ready == 1 && master\_valid == 1 ) ? ADDR\_IN : IDLE ) : ( (mode == 1) ? DATA\_IN : READ ) );

41.         DATA\_IN: next\_state = ( (counter < ADDR\_WIDTH+DATA\_WIDTH) ? ( ( port\_ready == 1 && master\_valid == 1 ) ? DATA\_IN : IDLE ) : WRITE );

42.         WRITE: next\_state = IDLE;

43.         READ: next\_state = ( (read\_counter == READ\_LATENCY+1) ? SEND : ( (SPLIT\_EN ? SPLIT : READ) ) );

44.         SPLIT: next\_state = ( (read\_counter == READ\_LATENCY+1) ? SEND : SPLIT);

45.         SEND: next\_state = ( (counter < DATA\_WIDTH) ? SEND : IDLE );

46.         default: next\_state = IDLE;

47.     endcase

48. end

49.

50. always\_ff@(posedge clk or negedge rstn) begin : STATE\_SEQUENCER

51.     if (!rstn) state <= IDLE;

52.     else state <= next\_state;

53. end

54.

55. // OUTPUT DECODER

56. assign port\_ready = (state == ADDR\_IN) | (state == DATA\_IN);

57. assign port\_valid = (state == SEND);

58. assign split = (state == SPLIT);

59. assign rd\_bus = ram\_in[DATA\_WIDTH-1-counter];

60.

61. always\_ff@(posedge clk) begin : OUTPUT\_DECODER

62.     case (state)

63.         IDLE: begin

64.             counter <= 0;

65.             addr\_in <= 0;

66.             data\_in <= 0;

67.             read\_counter <= 0;

68.         end

69.

70.         ADDR\_IN: begin

71.             addr\_in[ADDR\_WIDTH-1-counter] <= wr\_bus;

72.             counter <= counter + 1;

73.         end

74.

75.         DATA\_IN: begin

76.             data\_in[DATA\_WIDTH-1+ADDR\_WIDTH-counter] <= wr\_bus;

77.             counter <= counter + 1;

78.         end

79.

80.         READ: begin

81.             counter <= 0;

82.             read\_counter <= read\_counter + 1;

83.         end

84.

85.         SPLIT: begin

86.             read\_counter <= read\_counter + 1;

87.         end

88.

89.         SEND: begin

90.             if (master\_ready == 1) counter <= counter + 1;

91.         end

92.     endcase

93. end

94. endmodule

## Arbiter

1. module arbiter#(

2.     parameter  bus\_priority = 0

3. )(

4.     input   logic       clk,

5.     input   logic       rstn,

6.

7.     // connections to master 1

8.     input   logic       m1\_mode,

9.     output  logic       m1\_rd\_bus,

10.     input   logic       m1\_wr\_bus,

11.     output  logic       m1\_ack,

12.     input   logic       m1\_master\_valid,

13.     output  logic       m1\_slave\_ready,

14.     input   logic       m1\_master\_ready,

15.     output  logic       m1\_slave\_valid,

16.     input   logic       m1\_breq,

17.     output  logic       m1\_bgrant,

18.

19.     // connections to master 2

20.     input   logic       m2\_mode,

21.     output  logic       m2\_rd\_bus,

22.     input   logic       m2\_wr\_bus,

23.     output  logic       m2\_ack,

24.     input   logic       m2\_master\_valid,

25.     output  logic       m2\_slave\_ready,

26.     input   logic       m2\_master\_ready,

27.     output  logic       m2\_slave\_valid,

28.     input   logic       m2\_breq,

29.     output  logic       m2\_bgrant,

30.

31.     // connections to slave 1

32.     output  logic       s1\_mode,

33.     output  logic       s1\_wr\_bus,

34.     output  logic       s1\_master\_valid,

35.     output  logic       s1\_master\_ready,

36.     input   logic       s1\_rd\_bus,

37.     input   logic       s1\_slave\_ready,

38.     input   logic       s1\_slave\_valid,

39.

40.     // connections to slave 2

41.     output  logic       s2\_mode,

42.     output  logic       s2\_wr\_bus,

43.     output  logic       s2\_master\_valid,

44.     output  logic       s2\_master\_ready,

45.     input   logic       s2\_rd\_bus,

46.     input   logic       s2\_slave\_ready,

47.     input   logic       s2\_slave\_valid,

48.

49.     // connections to slave 3

50.     output  logic       s3\_mode,

51.     output  logic       s3\_wr\_bus,

52.     output  logic       s3\_master\_valid,

53.     output  logic       s3\_master\_ready,

54.     input   logic       s3\_rd\_bus,

55.     input   logic       s3\_slave\_ready,

56.     input   logic       s3\_slave\_valid,

57.

58.     // connections to bus bridge

59.     output  logic       bb\_mode,

60.     output  logic       bb\_wr\_bus,

61.     output  logic       bb\_master\_valid,

62.     output  logic       bb\_master\_ready,

63.     input   logic       bb\_rd\_bus,

64.     input   logic       bb\_slave\_ready,

65.     input   logic       bb\_slave\_valid,

66.

67.     input   logic       slave\_split,

68.     output  logic       m1\_split,

69.     output  logic       m2\_split

70. );

71.

72.     enum logic[2:0] {IDLE, ADDR, CONNECTED, CLEAN, REQ1, REQ2, REQ12, SPLIT} state, next\_state;

73.     enum logic[1:0] {S1, S2, S3, BB} slave;

74.     enum logic[1:0] {NONE, M1, M2} split\_owner;

75.

76.     logic[4:0]  t\_addr;

77.     logic[4:0]  t\_count;

78.     logic       t\_slave\_ready;

79.     logic       bus\_owner;

80.

81.     logic       mode;

82.     logic       rd\_bus;

83.     logic       wr\_bus;

84.     logic       ack;

85.     logic       master\_valid;

86.     logic       slave\_ready;

87.     logic       master\_ready;

88.     logic       slave\_valid;

89.     logic       breq;

90.

91.     always\_comb begin : NEXT\_STATE\_LOGIC

92.         case (state)

93.             IDLE:               begin

94.                                     if (split\_owner == NONE)

95.                                         next\_state = ( (m1\_breq == 0) ? ( (m2\_breq == 0) ? IDLE : REQ2 ) : ( (m2\_breq == 0) ? REQ1 : REQ12 ) );

96.                                     else if (slave\_split == 0)

97.                                         next\_state = CONNECTED;

98.                                     else

99.                                         next\_state = ( (m1\_breq == 0 || split\_owner == M1) ? ( (m2\_breq == 0 || split\_owner == M2) ? IDLE : REQ2 ) : ( (m2\_breq == 0 || split\_owner == M2) ? REQ1 : REQ12 ) );

100.                                 end

101.             REQ1:               next\_state = ( (m1\_breq == 0) ? IDLE : (m1\_master\_valid) ? ADDR : REQ1 );

102.             REQ2:               next\_state = ( (m2\_breq == 0) ? IDLE : (m2\_master\_valid) ? ADDR : REQ2 );

103.             REQ12:              next\_state = ( (m1\_breq == 0 || m2\_breq == 0) ? IDLE : (m1\_master\_valid) ? ADDR : REQ12 );

104.             ADDR:               next\_state = (t\_count == 5 && master\_valid) ? (ack ? CONNECTED : CLEAN) : ADDR;

105.             CONNECTED:          next\_state = (breq == 0) ? CLEAN : ((slave\_split && (t\_addr[4:1] == 4'b0001)) ? SPLIT : CONNECTED);

106.             SPLIT:              next\_state = IDLE;

107.             CLEAN:              next\_state = IDLE;

108.             default:            next\_state = IDLE;

109.         endcase

110.     end

111.

112.     always\_ff @(posedge clk or negedge rstn) begin : STATE\_SEQUENCER

113.         state <= !rstn ? IDLE : next\_state;

114.     end

115.

116.     assign slave\_ready = ack ? t\_slave\_ready : state == ADDR;

117.     assign m1\_bgrant = !bus\_owner;

118.     assign m2\_bgrant = bus\_owner;

119.     assign m1\_split = (split\_owner == M1 ? slave\_split : 0);

120.     assign m2\_split = (split\_owner == M2 ? slave\_split : 0);

121.

122.     always\_comb begin

123.         if (bus\_priority == 0) begin

124.             case (bus\_owner)

125.                 1'b1: begin

126.                     mode           = m2\_mode;

127.                     m2\_rd\_bus      = rd\_bus;

128.                     wr\_bus         = m2\_wr\_bus;

129.                     m2\_ack         = ack;

130.                     master\_valid   = m2\_master\_valid;

131.                     m2\_slave\_ready = slave\_ready;

132.                     master\_ready   = m2\_master\_ready;

133.                     m2\_slave\_valid = slave\_valid;

134.                     breq           = m2\_breq;

135.

136.                     m1\_rd\_bus      = 0;

137.                     m1\_ack         = 0;

138.                     m1\_slave\_ready = 0;

139.                     m1\_slave\_valid = 0;

140.                 end

141.

142.                 default: begin

143.                     mode           = m1\_mode;

144.                     m1\_rd\_bus      = rd\_bus;

145.                     wr\_bus         = m1\_wr\_bus;

146.                     m1\_ack         = ack;

147.                     master\_valid   = m1\_master\_valid;

148.                     m1\_slave\_ready = slave\_ready;

149.                     master\_ready   = m1\_master\_ready;

150.                     m1\_slave\_valid = slave\_valid;

151.                     breq           = m1\_breq;

152.

153.                     m2\_rd\_bus      = 0;

154.                     m2\_ack         = 0;

155.                     m2\_slave\_ready = 0;

156.                     m2\_slave\_valid = 0;

157.                 end

158.             endcase

159.         end else begin

160.             case (bus\_owner)

161.                 1'b0: begin

162.                     mode           = m1\_mode;

163.                     m1\_rd\_bus      = rd\_bus;

164.                     wr\_bus         = m1\_wr\_bus;

165.                     m1\_ack         = ack;

166.                     master\_valid   = m1\_master\_valid;

167.                     m1\_slave\_ready = slave\_ready;

168.                     master\_ready   = m1\_master\_ready;

169.                     m1\_slave\_valid = slave\_valid;

170.                     breq           = m1\_breq;

171.

172.                     m2\_rd\_bus      = 0;

173.                     m2\_ack         = 0;

174.                     m2\_slave\_ready = 0;

175.                     m2\_slave\_valid = 0;

176.                 end

177.

178.                 default: begin

179.                     mode           = m2\_mode;

180.                     m2\_rd\_bus      = rd\_bus;

181.                     wr\_bus         = m2\_wr\_bus;

182.                     m2\_ack         = ack;

183.                     master\_valid   = m2\_master\_valid;

184.                     m2\_slave\_ready = slave\_ready;

185.                     master\_ready   = m2\_master\_ready;

186.                     m2\_slave\_valid = slave\_valid;

187.                     breq           = m2\_breq;

188.

189.                     m1\_rd\_bus      = 0;

190.                     m1\_ack         = 0;

191.                     m1\_slave\_ready = 0;

192.                     m1\_slave\_valid = 0;

193.                 end

194.             endcase

195.         end

196.     end

197.

198.     always\_comb begin : OUTPUT\_LOGIC

199.         case (t\_addr[4:3])

200.             2'b11: begin

201.                 slave = BB;

202.                 ack = t\_count > 1;

203.             end

204.

205.             2'b00: begin

206.                 case (t\_addr[2:1])

207.                     2'b01: begin

208.                         slave = S2;

209.                         ack = t\_count > 3;

210.                     end

211.                     2'b10: begin

212.                         slave = S3;

213.                         ack = t\_count > 3;

214.                     end

215.                     2'b00: begin

216.                         case (t\_addr[0])

217.                             1'b0: begin

218.                                 slave = S1;

219.                                 ack = t\_count > 4;

220.                             end

221.                             1'b1: begin

222.                                 slave = S1;

223.                                 ack = 0;

224.                             end

225.                         endcase

226.                     end

227.                     default: begin

228.                         slave = S1;

229.                         ack = 0;

230.                     end

231.                 endcase

232.             end

233.             default: begin

234.                 slave = S1;

235.                 ack = 0;

236.             end

237.         endcase

238.

239.         case (slave)

240.             S1: begin

241.                 s1\_mode         = mode;

242.                 s1\_wr\_bus       = wr\_bus;

243.                 s1\_master\_valid = master\_valid && ack;

244.                 s1\_master\_ready = master\_ready;

245.                 t\_slave\_ready   = s1\_slave\_ready;

246.                 slave\_valid     = s1\_slave\_valid;

247.                 rd\_bus          = s1\_rd\_bus;

248.

249.                 s2\_mode         = 0;

250.                 s2\_wr\_bus       = 0;

251.                 s2\_master\_valid = 0;

252.                 s2\_master\_ready = 0;

253.

254.                 s3\_mode         = 0;

255.                 s3\_wr\_bus       = 0;

256.                 s3\_master\_valid = 0;

257.                 s3\_master\_ready = 0;

258.

259.                 bb\_mode         = 0;

260.                 bb\_wr\_bus       = 0;

261.                 bb\_master\_valid = 0;

262.                 bb\_master\_ready = 0;

263.             end

264.             S2: begin

265.                 s1\_mode         = 0;

266.                 s1\_wr\_bus       = 0;

267.                 s1\_master\_valid = 0;

268.                 s1\_master\_ready = 0;

269.

270.                 s2\_mode         = mode;

271.                 s2\_wr\_bus       = wr\_bus;

272.                 s2\_master\_valid = master\_valid && ack;

273.                 s2\_master\_ready = master\_ready;

274.                 t\_slave\_ready  = s2\_slave\_ready;

275.                 slave\_valid  = s2\_slave\_valid;

276.                 rd\_bus       = s2\_rd\_bus;

277.

278.                 s3\_mode         = 0;

279.                 s3\_wr\_bus       = 0;

280.                 s3\_master\_valid = 0;

281.                 s3\_master\_ready = 0;

282.

283.                 bb\_mode         = 0;

284.                 bb\_wr\_bus       = 0;

285.                 bb\_master\_valid = 0;

286.                 bb\_master\_ready = 0;

287.             end

288.             S3: begin

289.                 s1\_mode         = 0;

290.                 s1\_wr\_bus       = 0;

291.                 s1\_master\_valid = 0;

292.                 s1\_master\_ready = 0;

293.

294.                 s2\_mode         = 0;

295.                 s2\_wr\_bus       = 0;

296.                 s2\_master\_valid = 0;

297.                 s2\_master\_ready = 0;

298.

299.                 s3\_mode         = mode;

300.                 s3\_wr\_bus       = wr\_bus;

301.                 s3\_master\_valid = master\_valid && ack;

302.                 s3\_master\_ready = master\_ready;

303.                 t\_slave\_ready  = s3\_slave\_ready;

304.                 slave\_valid  = s3\_slave\_valid;

305.                 rd\_bus       = s3\_rd\_bus;

306.

307.                 bb\_mode         = 0;

308.                 bb\_wr\_bus       = 0;

309.                 bb\_master\_valid = 0;

310.                 bb\_master\_ready = 0;

311.             end

312.             BB: begin

313.                 s1\_mode         = 0;

314.                 s1\_wr\_bus       = 0;

315.                 s1\_master\_valid = 0;

316.                 s1\_master\_ready = 0;

317.

318.                 s2\_mode         = 0;

319.                 s2\_wr\_bus       = 0;

320.                 s2\_master\_valid = 0;

321.                 s2\_master\_ready = 0;

322.

323.                 s3\_mode         = 0;

324.                 s3\_wr\_bus       = 0;

325.                 s3\_master\_valid = 0;

326.                 s3\_master\_ready = 0;

327.

328.                 bb\_mode         = mode;

329.                 bb\_wr\_bus       = wr\_bus;

330.                 bb\_master\_valid = master\_valid && ack;

331.                 bb\_master\_ready = master\_ready;

332.                 t\_slave\_ready  = bb\_slave\_ready;

333.                 slave\_valid  = bb\_slave\_valid;

334.                 rd\_bus       = bb\_rd\_bus;

335.             end

336.         endcase

337.     end

338.

339.     always\_ff @(posedge clk or negedge rstn) begin : REG\_LOGIC

340.         if (!rstn) begin

341.             t\_count     <= 0;

342.             t\_addr      <= 0;

343.             bus\_owner   <= 0;

344.             split\_owner <= NONE;

345.         end else begin

346.             case (state)

347.                 IDLE: begin

348.                     if (split\_owner != NONE && slave\_split == 0) begin

349.                         bus\_owner <= (split\_owner == M1 ? 0 : 1);

350.                         t\_addr[4:1] <= 4'b0001;

351.                     end

352.                 end

353.

354.                 REQ1: begin

355.                     bus\_owner <= 0;

356.                 end

357.

358.                 REQ2: begin

359.                     bus\_owner <= 1;

360.                 end

361.

362.                 REQ12: begin

363.                     bus\_owner <= bus\_priority;

364.                 end

365.

366.                 ADDR: if (master\_valid) begin

367.                     t\_count <= t\_count + 1;

368.                     t\_addr[4 - t\_count] <= wr\_bus;

369.                 end

370.

371.                 CONNECTED: begin

372.                     t\_count <= t\_count + 1;

373.                 end

374.

375.                 SPLIT: begin

376.                     split\_owner <= (bus\_owner == 0) ? M1 : M2;

377.                 end

378.

379.                 CLEAN: begin

380.                     t\_count <= 0;

381.                     t\_addr  <= 0;

382.

383.                     if (bus\_owner == 0 && split\_owner == M1) split\_owner <= NONE;

384.                     else if (bus\_owner == 1 && split\_owner == M2) split\_owner <= NONE;

385.                 end

386.             endcase

387.         end

388.     end

389. endmodule

## Master Bus Bridge

1. module bb\_master\_port (

2.     input   logic       clk,

3.     input   logic       rstn,

4.     // connections to bus

5.     output  logic       mode,

6.     input   logic       rd\_bus,

7.     output  logic       wr\_bus,

8.     input   logic       ack,

9.     output  logic       master\_valid,

10.     input   logic       slave\_ready,

11.     output  logic       master\_ready,

12.     input   logic       slave\_valid,

13.     output  logic       breq,

14.     input   logic       bgrant,

15.     input   logic       split,

16.

17.     // connections to master

18.     input   logic[24:0] fifo\_data\_in,

19.     output  logic[7:0]  uart\_register\_out,

20.     output  logic       m\_out\_valid,

21.     input   logic       fifo\_empty,

22.     output  logic       fifo\_deq

23. );

24.     enum logic[3:0] {IDLE, REQ, ADDR\_1, ADDR\_2, WR\_DATA, RD\_DATA, CLEAN, SPLIT, TIMEOUT\_STATE, FETCH, DEQ} state, next\_state;

25.     localparam TIMEOUT = 64;

26.

27.     logic[3:0]  t\_count;

28.     logic[$clog2(TIMEOUT)-1:0] timeout;

29.     logic[7:0]  t\_wr\_data;

30.     logic[7:0]  t\_rd\_data;

31.     logic[15:0] t\_addr;

32.     logic       t\_mode;

33.

34.     always\_comb begin : NEXT\_STATE\_LOGIC

35.         case (state)

36.             IDLE:               next\_state = !fifo\_empty ? DEQ : IDLE;

37.             DEQ:                next\_state = FETCH;

38.             FETCH:              next\_state = REQ;

39.             REQ:                next\_state = bgrant ? ADDR\_1 : REQ;

40.             ADDR\_1:             next\_state = timeout != TIMEOUT - 1 ? ((t\_count == 5 & slave\_ready) ? (ack ? ADDR\_2 : CLEAN) : ADDR\_1) : TIMEOUT\_STATE;

41.             TIMEOUT\_STATE:      next\_state = REQ;

42.             ADDR\_2:             next\_state = (t\_count == 15 & slave\_ready) ? (t\_mode ? WR\_DATA : RD\_DATA) : ADDR\_2;

43.             WR\_DATA:            next\_state = (t\_count == 7  & slave\_ready) ? IDLE : WR\_DATA;

44.             RD\_DATA:            next\_state = split == 0 ? ((t\_count == 7 & slave\_valid) ? CLEAN : RD\_DATA) : SPLIT;

45.             SPLIT:              next\_state = split ? SPLIT : RD\_DATA;

46.             CLEAN:              next\_state = IDLE;

47.             default:            next\_state = IDLE;

48.         endcase

49.     end

50.

51.     always\_ff @(posedge clk or negedge rstn) begin : STATE\_SEQUENCER

52.         state <= !rstn ? IDLE : next\_state;

53.     end

54.

55.     always\_comb begin : OUTPUT\_LOGIC

56.         wr\_bus  = state == WR\_DATA ? t\_wr\_data[7] : t\_addr[15 - t\_count];

57.         mode    = t\_mode;

58.         master\_valid = state == ADDR\_2 || state == ADDR\_1 || state == WR\_DATA;

59.         master\_ready = state == RD\_DATA;

60.         uart\_register\_out = t\_rd\_data;

61.         m\_out\_valid = state == CLEAN & t\_count == 8;

62.         breq = state != IDLE && state != TIMEOUT\_STATE;

63.         fifo\_deq = state == DEQ;

64.     end

65.

66.     always\_ff @(posedge clk or negedge rstn) begin : REG\_LOGIC

67.         if (!rstn) begin

68.             t\_count   <= 0;

69.             t\_wr\_data <= 0;

70.             t\_rd\_data <= 0;

71.             t\_addr    <= 0;

72.             t\_mode    <= 0;

73.             timeout   <= 0;

74.         end else begin

75.             case (state)

76.                 FETCH: begin

77.                     t\_wr\_data <= fifo\_data\_in[7:0];

78.                     t\_addr    <= fifo\_data\_in[23:8];

79.                     t\_mode    <= fifo\_data\_in[24];

80.                 end

81.

82.                 REQ: begin

83.                     timeout <= 0;

84.                     t\_count <= 0;

85.                 end

86.

87.                 ADDR\_1:begin

88.                     timeout <= timeout + 1;

89.

90.                     if(slave\_ready) begin

91.                         t\_count <= t\_count + 1;

92.                     end

93.                 end

94.

95.                 ADDR\_2: if(slave\_ready) begin

96.                     t\_count <= t\_count + 1;

97.                 end

98.

99.                 WR\_DATA: if(slave\_ready) begin

100.                     t\_wr\_data <= t\_wr\_data << 1;

101.                     t\_count <= t\_count + 1;

102.                 end

103.

104.                 RD\_DATA: if(slave\_valid) begin

105.                     t\_rd\_data <= {t\_rd\_data[6:0], rd\_bus};

106.                     t\_count <= t\_count + 1;

107.                 end

108.

109.                 CLEAN: begin

110.                     t\_count <= 0;

111.                     t\_wr\_data <= 0;

112.                     t\_rd\_data <= 0;

113.                     t\_addr    <= 0;

114.                     t\_mode    <= 0;

115.                     timeout   <= 0;

116.                 end

117.             endcase

118.         end

119.     end

120. endmodule

## Slave Bus Bridge

1. module slave\_bus\_bridge#(

2.     parameter ADDR\_WIDTH = 16,

3.     parameter DATA\_WIDTH = 8,

4.     parameter SPLIT\_EN = 0

5. )(

6.     input logic mode, wr\_bus, master\_valid, master\_ready, rstn, clk, valid\_in,

7.     input logic [DATA\_WIDTH-1:0]uart\_register\_in,

8.     output logic [1+16+DATA\_WIDTH-1:0]uart\_register\_out,

9.     output logic rd\_bus, slave\_ready, slave\_valid, split, valid\_out

10. );

11.

12. // local parameters

13. localparam COUNTER\_LENGTH = $clog2(ADDR\_WIDTH+DATA\_WIDTH);

14. localparam NUM\_STATES = 8;

15. localparam STATE\_N\_BITS = $clog2(NUM\_STATES);

16.

17. // internal signals

18. logic [COUNTER\_LENGTH-1:0]counter;

19. logic [ADDR\_WIDTH-1:0]addr\_in;

20. logic [DATA\_WIDTH-1:0]data\_in;

21. logic port\_ready, port\_valid;

22.

23. // definition of states

24. enum logic[STATE\_N\_BITS-1:0] {IDLE, ADDR\_IN, DATA\_IN, WRITE, READ, SEND, SPLIT, SEND\_RD\_ADDR} state, next\_state;

25.

26. assign slave\_ready = port\_ready;

27. assign slave\_valid = port\_valid;

28.

29. always\_comb begin : NEXT\_STATE\_DECODER

30.     case (state)

31.         IDLE: next\_state = ( ( master\_valid == 1 ) ? ADDR\_IN : IDLE );

32.         ADDR\_IN: next\_state = ( (counter < ADDR\_WIDTH-1) ? ( (port\_ready == 1 && master\_valid == 1 ) ? ADDR\_IN : IDLE ) : ( (mode == 1) ? DATA\_IN : SEND\_RD\_ADDR ) );

33.         DATA\_IN: next\_state = ( (counter < ADDR\_WIDTH+DATA\_WIDTH) ? ( ( port\_ready == 1 && master\_valid == 1 ) ? DATA\_IN : IDLE ) : WRITE );

34.         WRITE: next\_state = IDLE;

35.         SEND\_RD\_ADDR: next\_state = READ;

36.         READ: next\_state = ( (valid\_in) ? SEND : ( (SPLIT\_EN ? SPLIT : READ) ) );

37.         SPLIT: next\_state = ( (valid\_in) ? SEND : SPLIT);

38.         SEND: next\_state = ( (counter < DATA\_WIDTH) ? SEND : IDLE );

39.         default: next\_state = IDLE;

40.     endcase

41. end

42.

43. always\_ff@(posedge clk or negedge rstn) begin : STATE\_SEQUENCER

44.     if (!rstn) state <= IDLE;

45.     else state <= next\_state;

46. end

47.

48. // OUTPUT DECODER

49. assign port\_ready = (state == ADDR\_IN) | (state == DATA\_IN);

50. assign port\_valid = (state == SEND);

51. assign split = (state == SPLIT);

52. assign rd\_bus = uart\_register\_in[DATA\_WIDTH-1-counter];

53.

54. always\_ff@(posedge clk) begin : OUTPUT\_DECODER

55.     case (state)

56.         IDLE: begin

57.             counter <= 0;

58.             addr\_in <= 0;

59.             data\_in <= 0;

60.             valid\_out <= 0;

61.         end

62.

63.         ADDR\_IN: begin

64.             addr\_in[ADDR\_WIDTH-1-counter] <= wr\_bus;

65.             counter <= counter + 1;

66.         end

67.

68.         DATA\_IN: begin

69.             data\_in[DATA\_WIDTH-1+ADDR\_WIDTH-counter] <= wr\_bus;

70.             counter <= counter + 1;

71.         end

72.

73.         WRITE: begin

74.             uart\_register\_out <= {mode, 2'b00, addr\_in, data\_in};

75.             valid\_out <= 1;

76.         end

77.

78.         SEND\_RD\_ADDR: begin

79.             uart\_register\_out <= {mode, 2'b00, addr\_in, 8'd0};

80.             valid\_out <= 1;

81.         end

82.

83.         READ: begin

84.             valid\_out <= 0;

85.             counter <= 0;

86.         end

87.

88.         SEND: begin

89.             if (master\_ready == 1) counter <= counter + 1;

90.         end

91.     endcase

92. end

93. endmodule

## FIFO

1. module FIFO #(parameter WIDTH = 32, parameter DEPTH = 16) (

2.     input logic clk,

3.     input logic rstn,

4.     input logic [WIDTH-1:0] data\_in,

5.     input logic enq,

6.     input logic deq,

7.     output logic [WIDTH-1:0] data\_out,

8.     output logic empty

9. );

10.

11. logic [DEPTH-1:0][WIDTH-1:0] memory; //packed array

12. logic [$clog2(DEPTH):0] rd\_ptr, wr\_ptr;

13. logic full;

14.

15. assign data\_out = memory[rd\_ptr];

16. assign empty = (rd\_ptr == wr\_ptr);

17. assign full = (rd\_ptr == wr\_ptr + 1);

18.

19. always\_ff @(posedge clk or negedge rstn) begin

20.     if(!rstn) begin

21.         rd\_ptr <= 0;

22.         wr\_ptr <= 0;

23.     end

24.

25.     else begin

26.         if (enq && !full) begin

27.             memory[wr\_ptr] <= data\_in;

28.             wr\_ptr <= wr\_ptr + 1;

29.         end

30.

31.         if (deq && !empty) begin

32.             rd\_ptr <= rd\_ptr + 1;

33.         end

34.     end

35. end

36.

37. endmodule