UNIVERSITY OF MORATUWA

Department of Electronic and Telecommunication Engineering

Faculty of Engineering

EN4020 - Advanced Digital Systems

System Bus Design

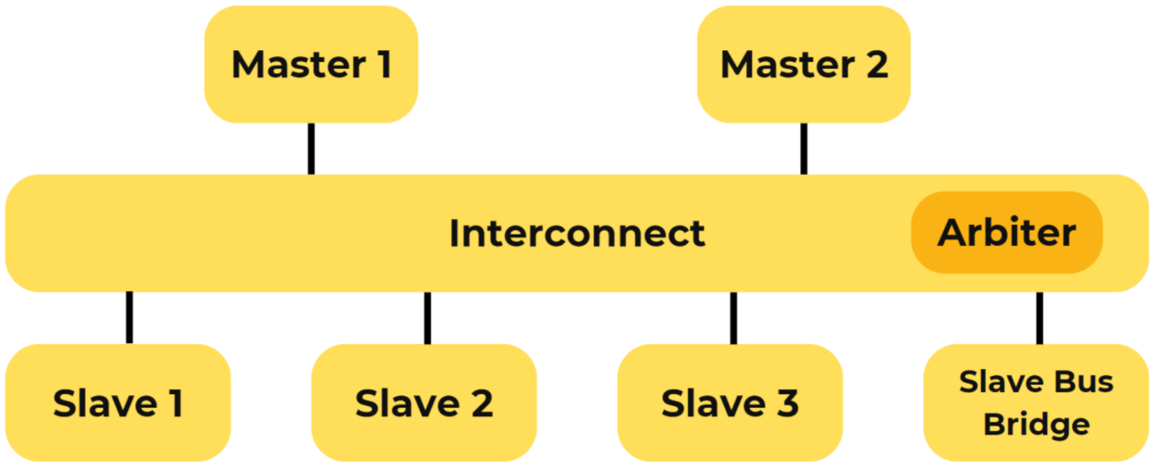
Group 01

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



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.

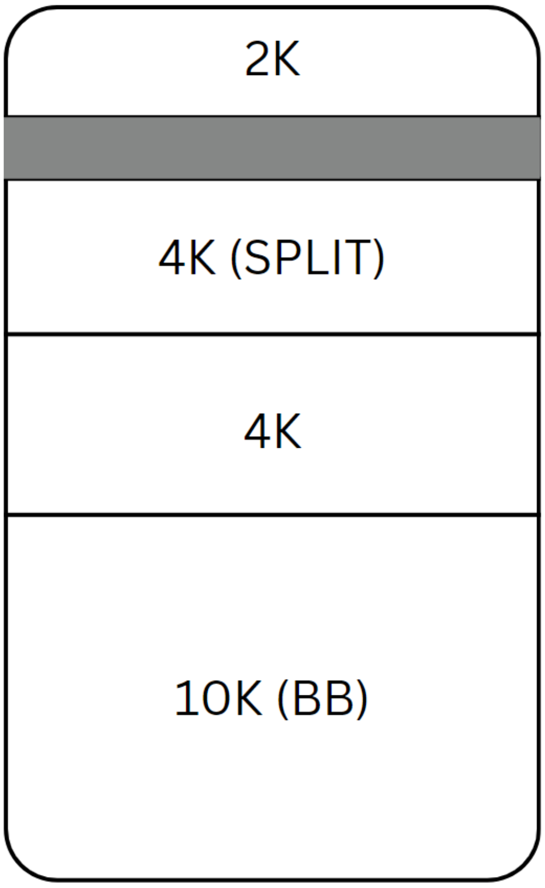
The 10 KB memory space for the bus bridge is made up in the following way.

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.

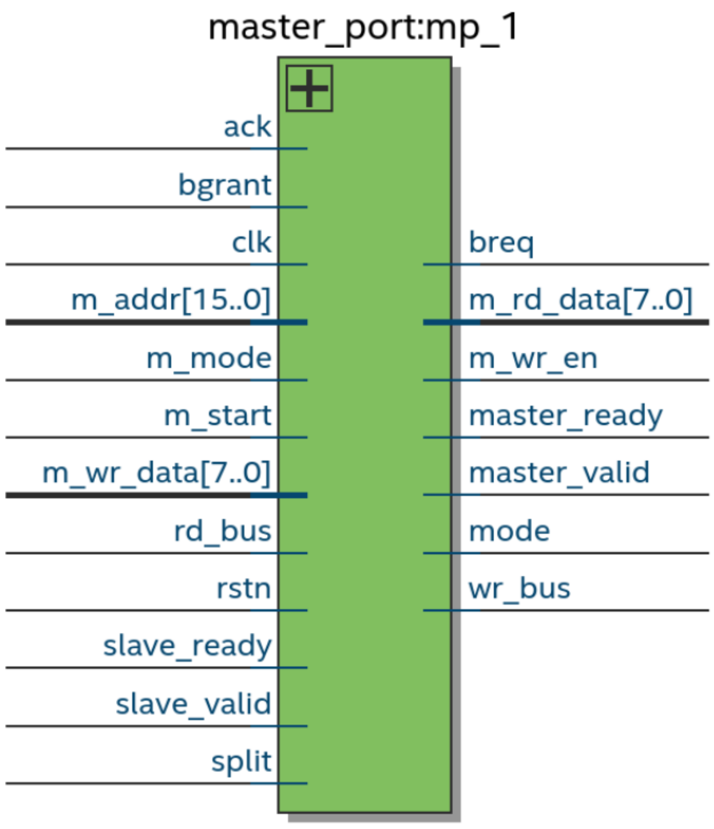
2. Address Allocation

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



3. Master

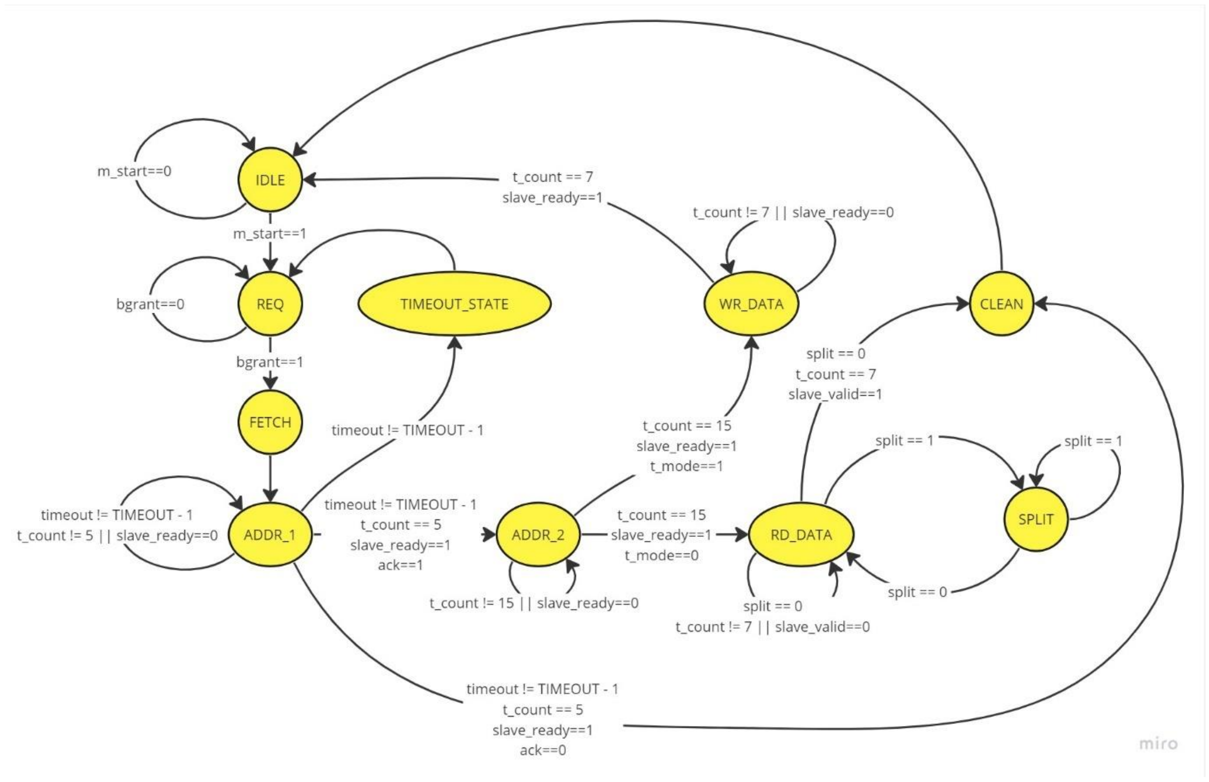
3.1. Block Diagram



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.

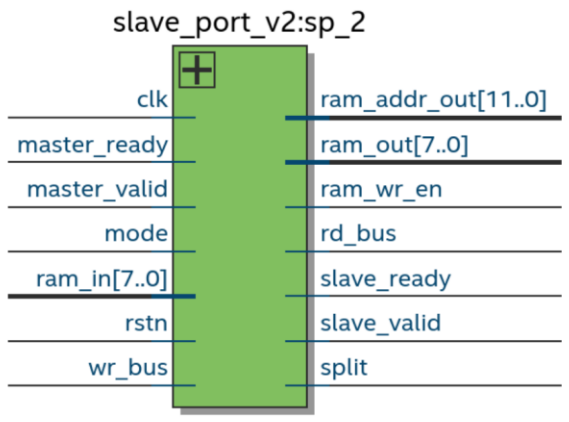
master_ready	Indicate master is ready to receive.
master_valid	Indicate data in wr_bus is valid.
mode	Read (mode = 0), write (mode = 1)
wr_bus	Serial data output bus.

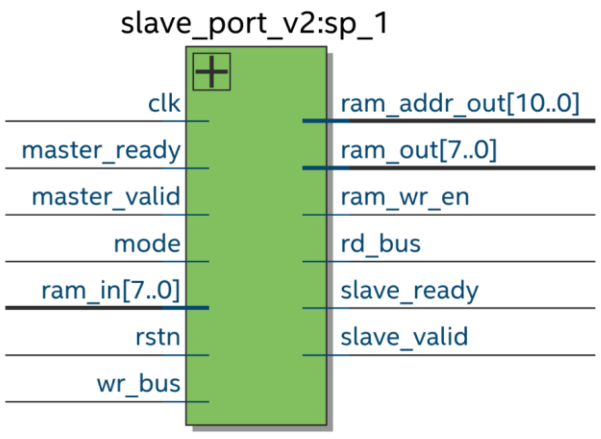
3.2. State Diagram



4. Slave

4.1. Block Diagram

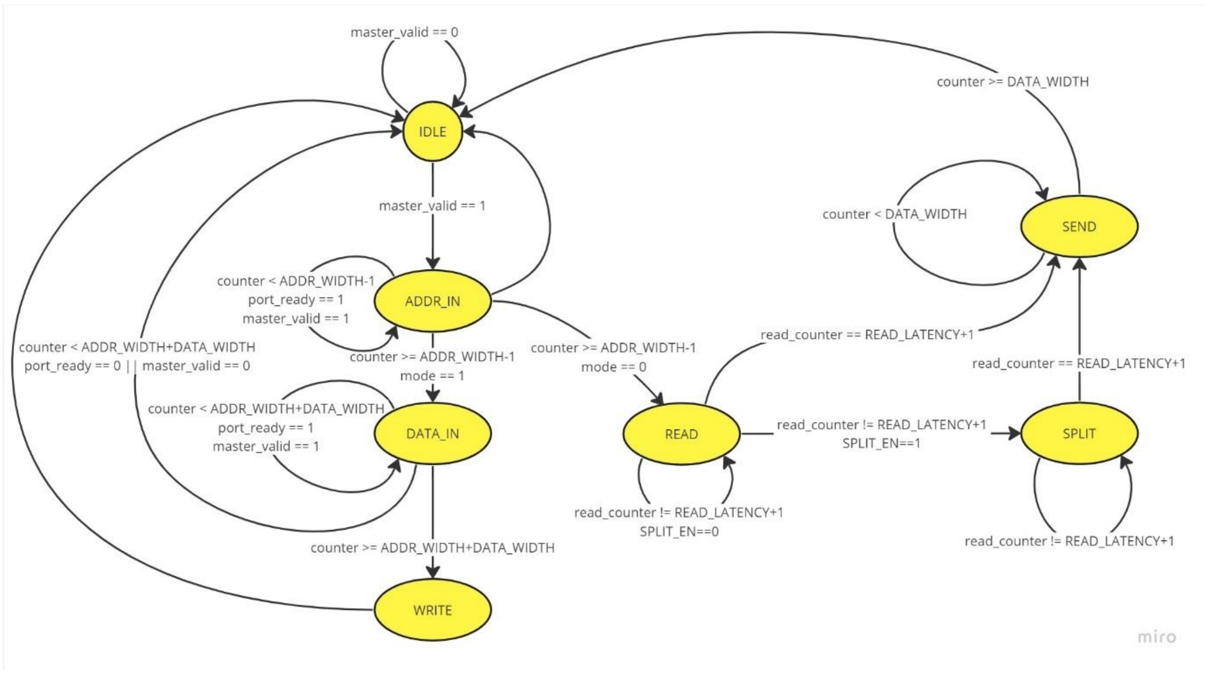




Input Pin	Description
clk	Clock signal
master_ready	Indicates master is ready to receive data
master_valid	Indicates the data in the wr_bus is valid
mode	Read (mode = 0), write (mode = 1)
ram_in[7..0]	Register to load the contents from slave BRAM memory
rstn	Active low reset signal
wr_bus	Serial data in port

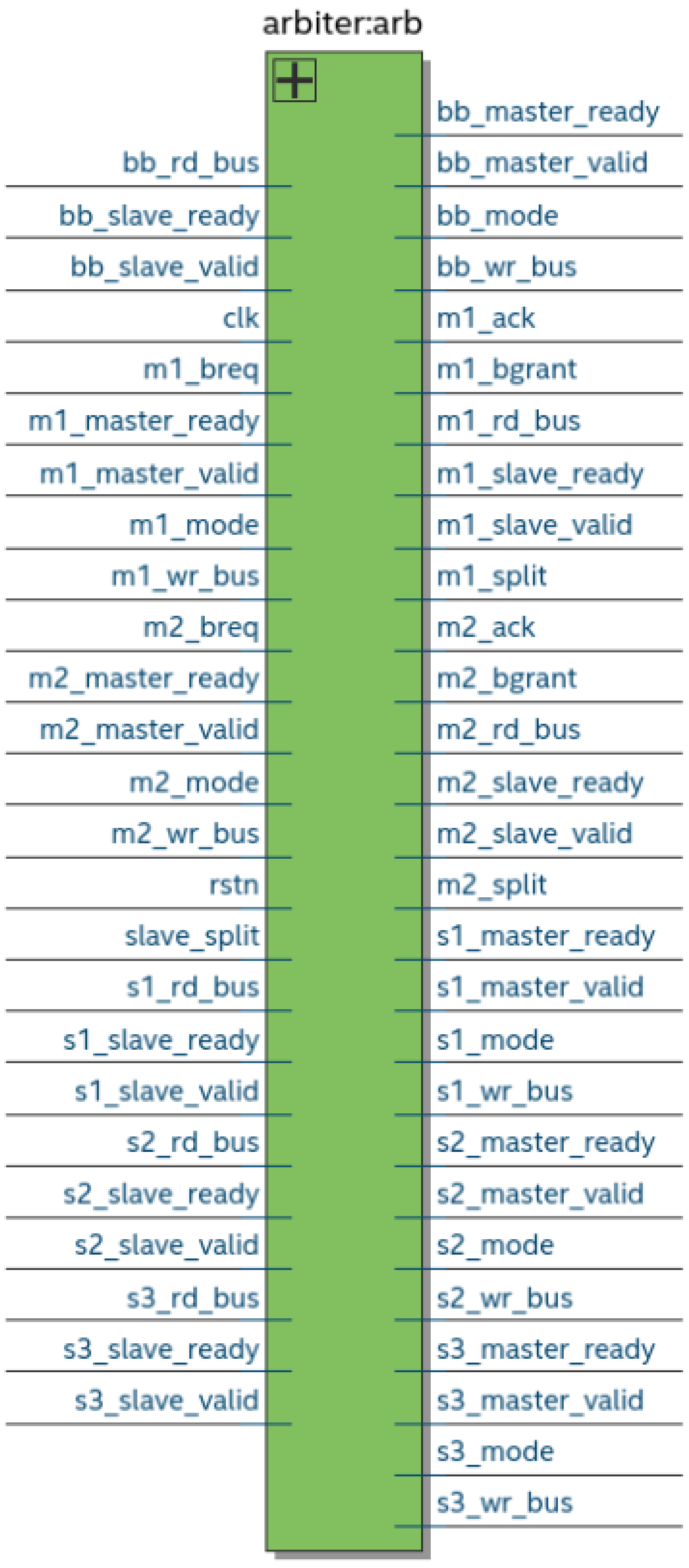
Output Pin	Description
ram_addr_out[11..0]	Register to address the BRAM memory
ram_out[7..0]	Register to send data to the BRAM memory
ram_wr_en	Register to enable writing to the BRAM memory
rd_bus	Serial data out port
slave_ready	Indicates the slave is ready to receive data
slave_valid	Indicates the data in the rd_bus is valid
split	Indicates to the arbiter to split the transaction

4.2. State Diagram



5. Arbiter

5.1. Block Diagram

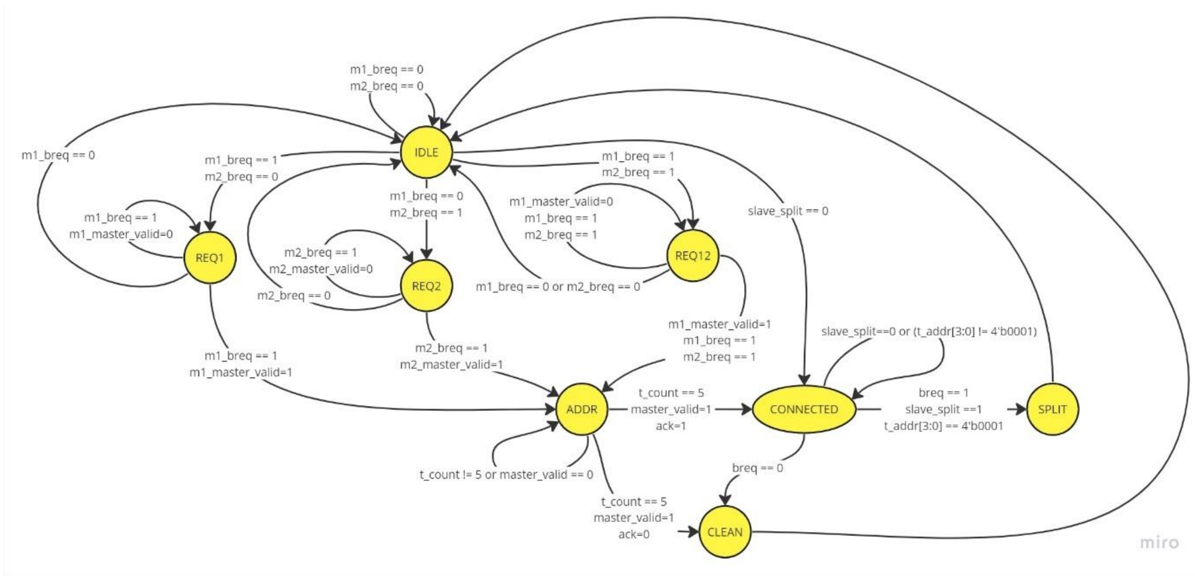


Input Pin	**Description**
bb_rd_bus	Bus bridge serial data out bus.
bb_slave_ready	Indicate bus bridge is ready to receive.
bb_salve_valid	Indicate data on bb_rd_bus is valid.
clk	Clock signal
m1_breq	Request signal of Master 1.
m1_master_ready	Master 1 is ready to receive.
m1_master_valid	Data on m1_wr_bus 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.

5.2. State Diagram



6. 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.

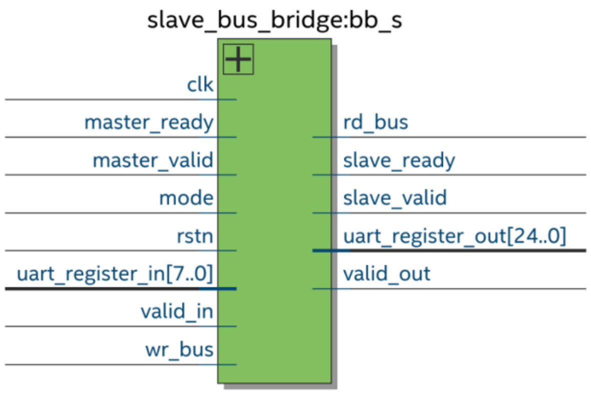
The master bus bridge is a master to our bus, and it interfaces with a slave bus bridge in the other team's bus via UART. This interface is used to receive transaction commands from masters and send data to masters for READ requests in the other team's bus.

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

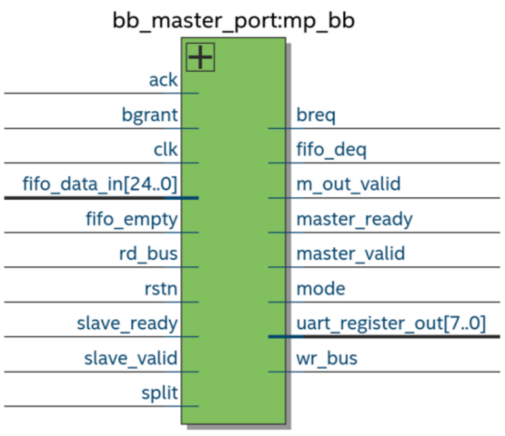
The exchange of data in a READ transaction is done through a UART port that facilitate transmission/reception of 8-bit width words per single transmission/reception. The entire 8-bit word is the data that need to be transmitted/received.

6.1. Slave 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

6.2. Master Bus Bridge

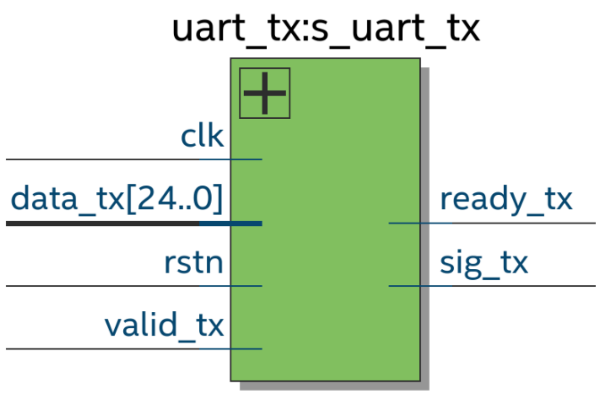


Input Pin	**Description**
ack	Acknowledgment coming from arbiter to indicate that sent address is valid.
bgrant	Indicate that master has access to bus.
clk	Clock signal
fifo_data_in[24..0]	Data coming from FIFO.
fifo_empty	Indicate that FIFO is empty.
rd_bus	Serial data input bus port.
rstn	Asynchronous 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.

6.3. Asynchronous RX TX

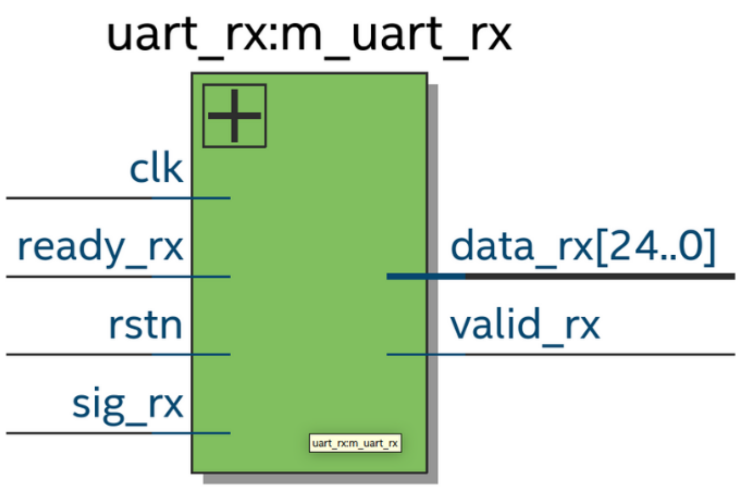
6.3.1. Transmitter



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

6.3.2. Receiver



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

7. Timing Analysis Report

7.1. Constrains

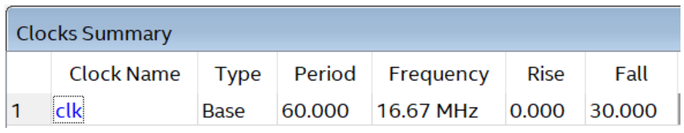
Constrains	Value (ns)
Clock Period	60
Clock Uncertainty	3 (5%)
Input Delay	24 (40%)
Output Delay	24 (40%)

7.2. Constrains File

```
#**************************************************************
# Time Information
#**************************************************************

set_time_format -unit ns -decimal_places 3

#**************************************************************
# Create Clock
#**************************************************************
create_clock -name {clk} -period 60.000 -waveform { 0.000 30.000 } [get_ports {clk}]

#**************************************************************
# Create Generated Clock
#**************************************************************

#**************************************************************
# Set Clock Latency
#**************************************************************

#**************************************************************
# Set Clock Uncertainty
#**************************************************************

set_clock_uncertainty -rise_from [get_clocks {clk}] -rise_to [get_clocks {clk}] -setup 3.000
set_clock_uncertainty -rise_from [get_clocks {clk}] -fall_to [get_clocks {clk}] -setup 3.000
set_clock_uncertainty -fall_from [get_clocks {clk}] -rise_to [get_clocks {clk}] -setup 3.000
set_clock_uncertainty -fall_from [get_clocks {clk}] -fall_to [get_clocks {clk}] -setup 3.000

#**************************************************************
# Set Input Delay
#**************************************************************

set_input_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {addr[0]}]
set_input_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {addr[1]}]
set_input_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {addr[2]}]
set_input_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {addr[3]}]
set_input_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {addr[4]}]
set_input_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {addr[5]}]
set_input_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {addr[6]}]
set_input_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {addr[7]}]
set_input_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {addr[8]}]
set_input_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {addr[9]}]
set_input_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {addr[10]}]
set_input_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {addr[11]}]
set_input_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {addr[12]}]
set_input_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {addr[13]}]
set_input_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {addr[14]}]
set_input_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {addr[15]}]
set_input_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {keysn[0]}]
set_input_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {keysn[1]}]
set_input_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {keysn[2]}]
set_input_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {keysn[3]}]
set_input_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {m_ready_rx}]
set_input_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {m_sig_rx}]
set_input_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {s_ready_rx}]
set_input_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {s_sig_rx}]

#**************************************************************
# Set Output Delay
#**************************************************************

set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {hex0[0]}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {hex0[1]}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {hex0[2]}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {hex0[3]}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {hex0[4]}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {hex0[5]}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {hex0[6]}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {hex1[0]}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {hex1[1]}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {hex1[2]}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {hex1[3]}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {hex1[4]}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {hex1[5]}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {hex1[6]}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {hex2[0]}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {hex2[1]}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {hex2[2]}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {hex2[3]}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {hex2[4]}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {hex2[5]}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {hex2[6]}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {hex3[0]}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {hex3[1]}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {hex3[2]}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {hex3[3]}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {hex3[4]}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {hex3[5]}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {hex3[6]}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {m1_ack_led}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {m1_master_ready_led}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {m1_master_valid_led}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {m1_mode_led}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {m1_slave_ready_led}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {m1_slave_valid_led}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {m2_mode_led}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {m_ready_tx}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {m_sig_tx}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {rstn_led}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {s1_master_ready_led}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {s1_master_valid_led}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {s1_slave_ready_led}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {s1_slave_valid_led}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {s_ready_tx}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  24.000 [get_ports {s_sig_tx}]

#**************************************************************
# Set Clock Groups
#**************************************************************

#**************************************************************
# Set False Path
#**************************************************************

#**************************************************************
# Set Multicycle Path
#**************************************************************

#**************************************************************
# Set Maximum Delay
#**************************************************************

#**************************************************************
# Set Minimum Delay
```

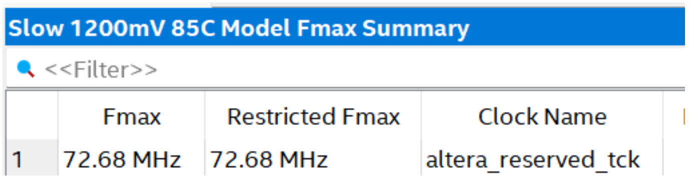
```
#**************************************************************

#**************************************************************
# Set Input Transition
#**************************************************************


```

7.3. Clocks Summary



Clocks Summary

	Clock Name	Type	Period	Frequency	Rise	Fall
1	clk	Base	60.000	16.67 MHz	0.000	30.000

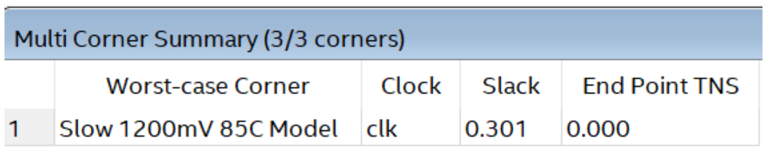


Slow 1200mV 85C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name
1	72.68 MHz	72.68 MHz	altera_reserved_tck

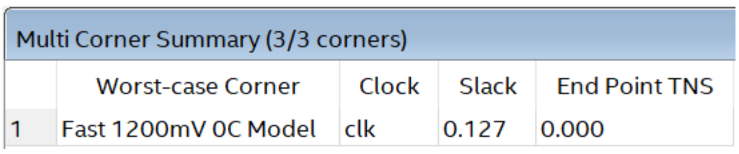
7.4. Summary (Setup)



Multi Corner Summary (3/3 corners)

	Worst-case Corner	Clock	Slack	End Point TNS
1	Slow 1200mV 85C Model	clk	0.301	0.000

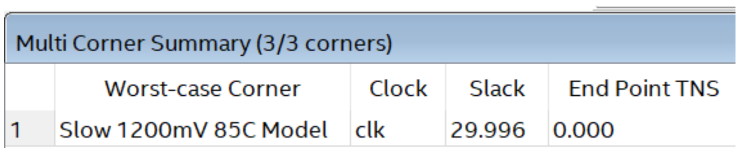
7.5. Summary (Hold)



Multi Corner Summary (3/3 corners)

	Worst-case Corner	Clock	Slack	End Point TNS
1	Fast 1200mV 0C Model	clk	0.127	0.000

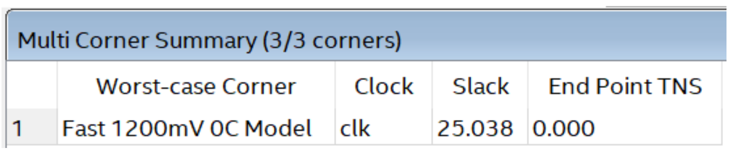
7.6. Summary (Recovery)



Multi Corner Summary (3/3 corners)

	Worst-case Corner	Clock	Slack	End Point TNS
1	Slow 1200mV 85C Model	clk	29.996	0.000

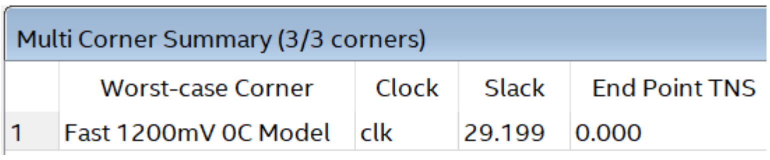
7.7. Summary (Removal)



Multi Corner Summary (3/3 corners)

	Worst-case Corner	Clock	Slack	End Point TNS
1	Fast 1200mV 0C Model	clk	25.038	0.000

7.8. Summary (Minimum Pulse Width)

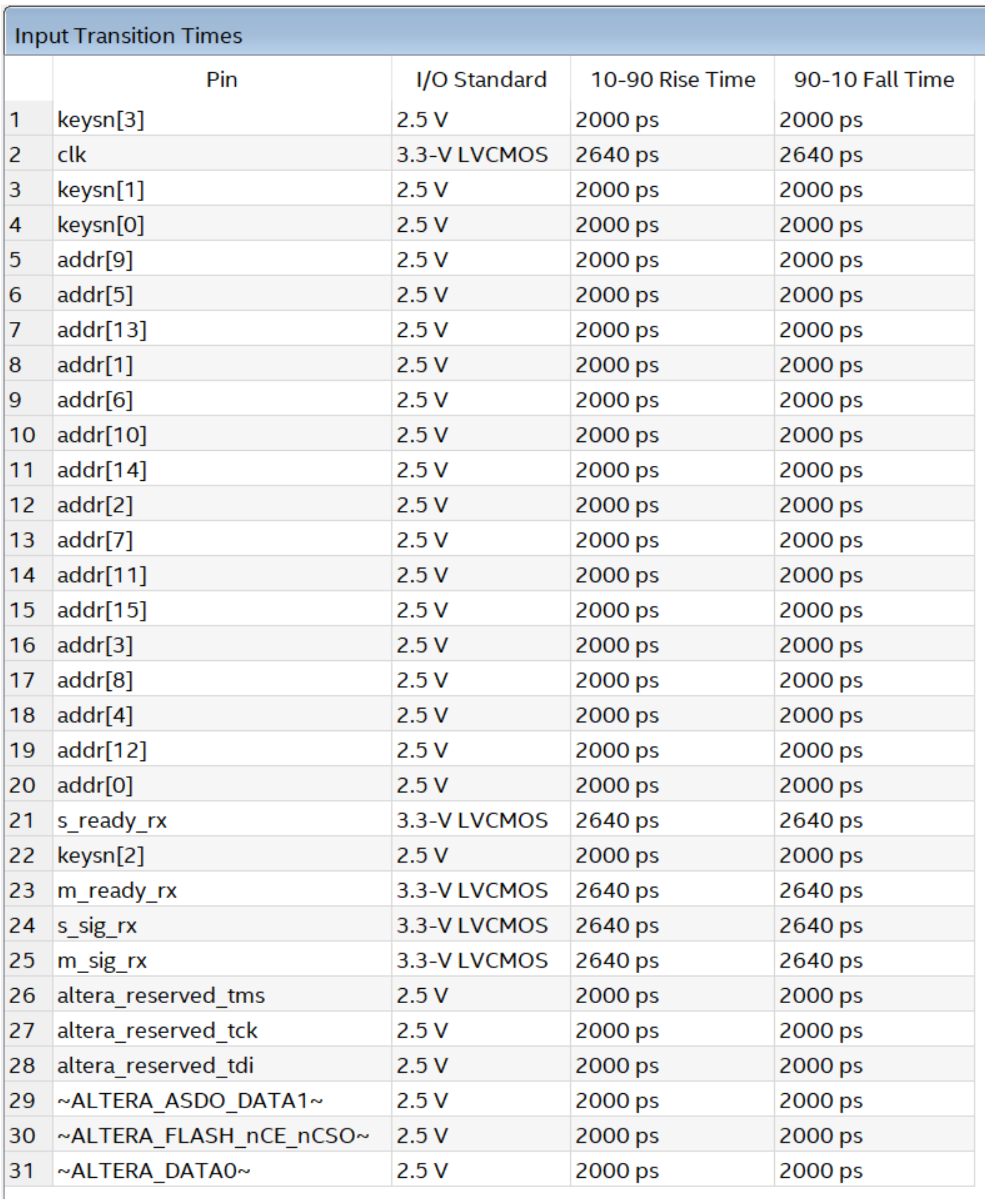


Multi Corner Summary (3/3 corners)

	Worst-case Corner	Clock	Slack	End Point TNS
1	Fast 1200mV 0C Model	clk	29.199	0.000

7.9. Advanced I/O Timing

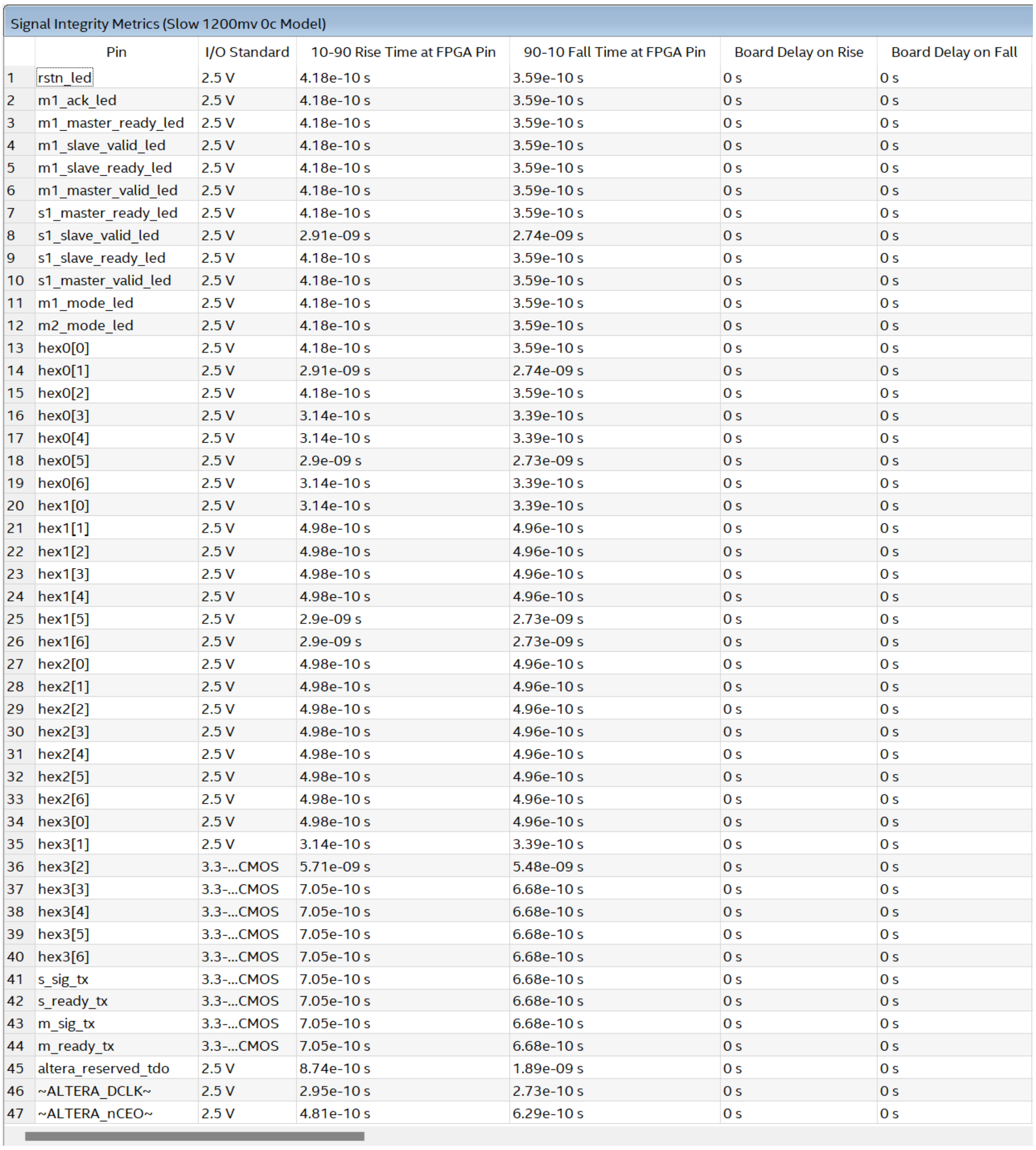
7.9.1. Input Transition Times



Input Transition Times

	Pin	I/O Standard	10-90 Rise Time	90-10 Fall Time
1	keysn[3]	2.5 V	2000 ps	2000 ps
2	clk	3.3-V LVCMOS	2640 ps	2640 ps
3	keysn[1]	2.5 V	2000 ps	2000 ps
4	keysn[0]	2.5 V	2000 ps	2000 ps
5	addr[9]	2.5 V	2000 ps	2000 ps
6	addr[5]	2.5 V	2000 ps	2000 ps
7	addr[13]	2.5 V	2000 ps	2000 ps
8	addr[1]	2.5 V	2000 ps	2000 ps
9	addr[6]	2.5 V	2000 ps	2000 ps
10	addr[10]	2.5 V	2000 ps	2000 ps
11	addr[14]	2.5 V	2000 ps	2000 ps
12	addr[2]	2.5 V	2000 ps	2000 ps
13	addr[7]	2.5 V	2000 ps	2000 ps
14	addr[11]	2.5 V	2000 ps	2000 ps
15	addr[15]	2.5 V	2000 ps	2000 ps
16	addr[3]	2.5 V	2000 ps	2000 ps
17	addr[8]	2.5 V	2000 ps	2000 ps
18	addr[4]	2.5 V	2000 ps	2000 ps
19	addr[12]	2.5 V	2000 ps	2000 ps
20	addr[0]	2.5 V	2000 ps	2000 ps
21	s_ready_rx	3.3-V LVCMOS	2640 ps	2640 ps
22	keysn[2]	2.5 V	2000 ps	2000 ps
23	m_ready_rx	3.3-V LVCMOS	2640 ps	2640 ps
24	s_sig_rx	3.3-V LVCMOS	2640 ps	2640 ps
25	m_sig_rx	3.3-V LVCMOS	2640 ps	2640 ps
26	altera_reserved_tms	2.5 V	2000 ps	2000 ps
27	altera_reserved_tck	2.5 V	2000 ps	2000 ps
28	altera_reserved_tdi	2.5 V	2000 ps	2000 ps
29	~ALTERA_ASDO_DATA1~	2.5 V	2000 ps	2000 ps
30	~ALTERA_FLASH_nCE_nCSO~	2.5 V	2000 ps	2000 ps
31	~ALTERA_DATA0~	2.5 V	2000 ps	2000 ps

7.9.2. Signal Integrity Metrics

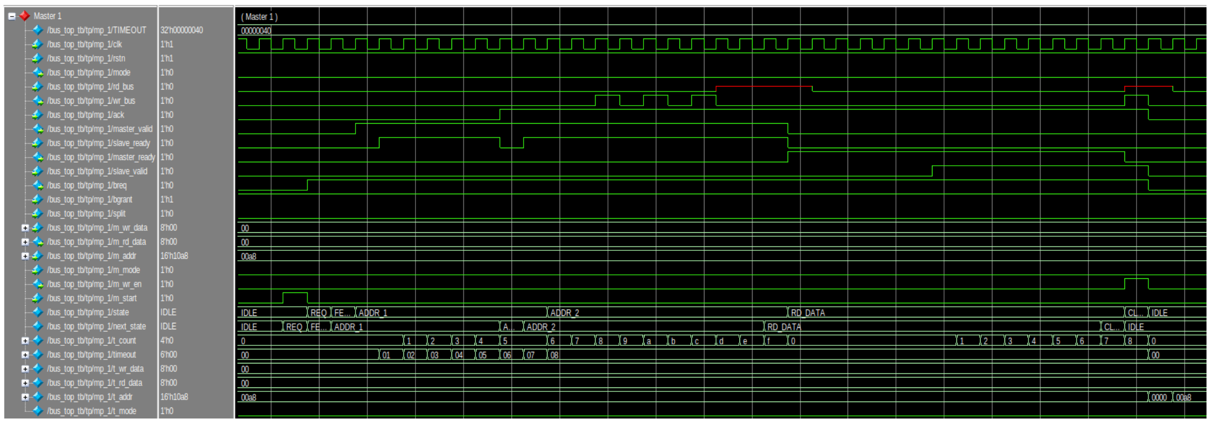


Signal Integrity Metrics (Slow 1200mv 0c Model)

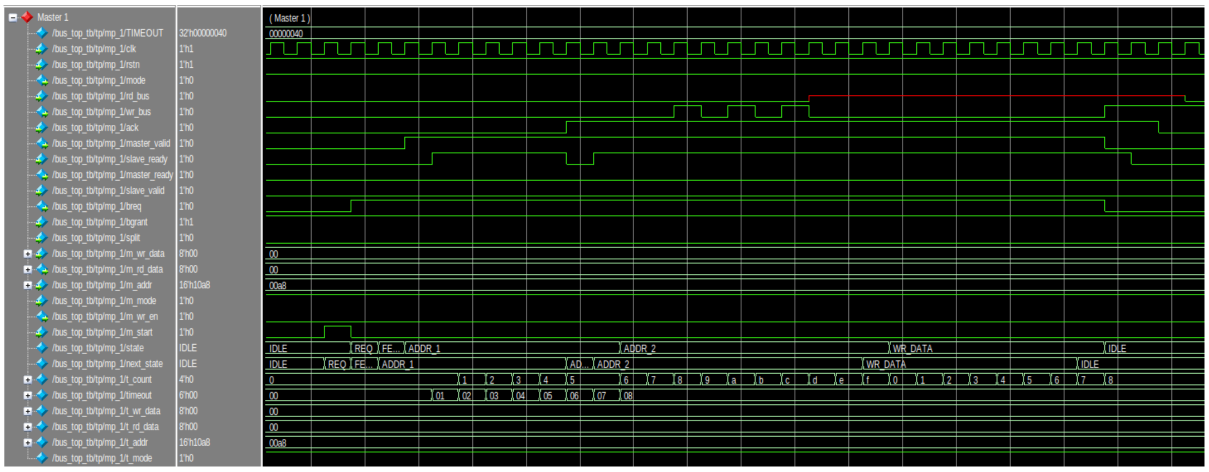
	Pin	I/O Standard	10-90 Rise Time at FPGA Pin	90-10 Fall Time at FPGA Pin	Board Delay on Rise	Board Delay on Fall
1	rstn_led	2.5 V	4.18e-10 s	3.59e-10 s	0 s	0 s
2	m1_ack_led	2.5 V	4.18e-10 s	3.59e-10 s	0 s	0 s
3	m1_master_ready_led	2.5 V	4.18e-10 s	3.59e-10 s	0 s	0 s
4	m1_slave_valid_led	2.5 V	4.18e-10 s	3.59e-10 s	0 s	0 s
5	m1_slave_ready_led	2.5 V	4.18e-10 s	3.59e-10 s	0 s	0 s
6	m1_master_valid_led	2.5 V	4.18e-10 s	3.59e-10 s	0 s	0 s
7	s1_master_ready_led	2.5 V	4.18e-10 s	3.59e-10 s	0 s	0 s
8	s1_slave_valid_led	2.5 V	2.91e-09 s	2.74e-09 s	0 s	0 s
9	s1_slave_ready_led	2.5 V	4.18e-10 s	3.59e-10 s	0 s	0 s
10	s1_master_valid_led	2.5 V	4.18e-10 s	3.59e-10 s	0 s	0 s
11	m1_mode_led	2.5 V	4.18e-10 s	3.59e-10 s	0 s	0 s
12	m2_mode_led	2.5 V	4.18e-10 s	3.59e-10 s	0 s	0 s
13	hex0[0]	2.5 V	4.18e-10 s	3.59e-10 s	0 s	0 s
14	hex0[1]	2.5 V	2.91e-09 s	2.74e-09 s	0 s	0 s
15	hex0[2]	2.5 V	4.18e-10 s	3.59e-10 s	0 s	0 s
16	hex0[3]	2.5 V	3.14e-10 s	3.39e-10 s	0 s	0 s
17	hex0[4]	2.5 V	3.14e-10 s	3.39e-10 s	0 s	0 s
18	hex0[5]	2.5 V	2.9e-09 s	2.73e-09 s	0 s	0 s
19	hex0[6]	2.5 V	3.14e-10 s	3.39e-10 s	0 s	0 s
20	hex1[0]	2.5 V	3.14e-10 s	3.39e-10 s	0 s	0 s
21	hex1[1]	2.5 V	4.98e-10 s	4.96e-10 s	0 s	0 s
22	hex1[2]	2.5 V	4.98e-10 s	4.96e-10 s	0 s	0 s
23	hex1[3]	2.5 V	4.98e-10 s	4.96e-10 s	0 s	0 s
24	hex1[4]	2.5 V	4.98e-10 s	4.96e-10 s	0 s	0 s
25	hex1[5]	2.5 V	2.9e-09 s	2.73e-09 s	0 s	0 s
26	hex1[6]	2.5 V	2.9e-09 s	2.73e-09 s	0 s	0 s
27	hex2[0]	2.5 V	4.98e-10 s	4.96e-10 s	0 s	0 s
28	hex2[1]	2.5 V	4.98e-10 s	4.96e-10 s	0 s	0 s
29	hex2[2]	2.5 V	4.98e-10 s	4.96e-10 s	0 s	0 s
30	hex2[3]	2.5 V	4.98e-10 s	4.96e-10 s	0 s	0 s
31	hex2[4]	2.5 V	4.98e-10 s	4.96e-10 s	0 s	0 s
32	hex2[5]	2.5 V	4.98e-10 s	4.96e-10 s	0 s	0 s
33	hex2[6]	2.5 V	4.98e-10 s	4.96e-10 s	0 s	0 s
34	hex3[0]	2.5 V	4.98e-10 s	4.96e-10 s	0 s	0 s
35	hex3[1]	2.5 V	3.14e-10 s	3.39e-10 s	0 s	0 s
36	hex3[2]	3.3-...CMOS	5.71e-09 s	5.48e-09 s	0 s	0 s
37	hex3[3]	3.3-...CMOS	7.05e-10 s	6.68e-10 s	0 s	0 s
38	hex3[4]	3.3-...CMOS	7.05e-10 s	6.68e-10 s	0 s	0 s
39	hex3[5]	3.3-...CMOS	7.05e-10 s	6.68e-10 s	0 s	0 s
40	hex3[6]	3.3-...CMOS	7.05e-10 s	6.68e-10 s	0 s	0 s
41	s_sig_tx	3.3-...CMOS	7.05e-10 s	6.68e-10 s	0 s	0 s
42	s_ready_tx	3.3-...CMOS	7.05e-10 s	6.68e-10 s	0 s	0 s
43	m_sig_tx	3.3-...CMOS	7.05e-10 s	6.68e-10 s	0 s	0 s
44	m_ready_tx	3.3-...CMOS	7.05e-10 s	6.68e-10 s	0 s	0 s
45	altera_reserved_tdo	2.5 V	8.74e-10 s	1.89e-09 s	0 s	0 s
46	~ALTERA_DCLK~	2.5 V	2.95e-10 s	2.73e-10 s	0 s	0 s
47	~ALTERA_nCEO~	2.5 V	4.81e-10 s	6.29e-10 s	0 s	0 s

8. Simulation results

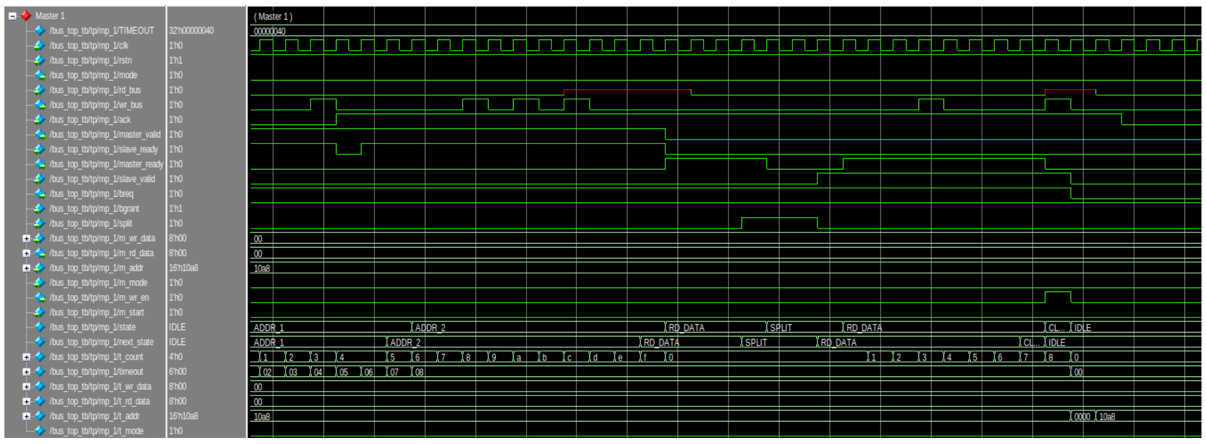
8.1. Master read transaction.



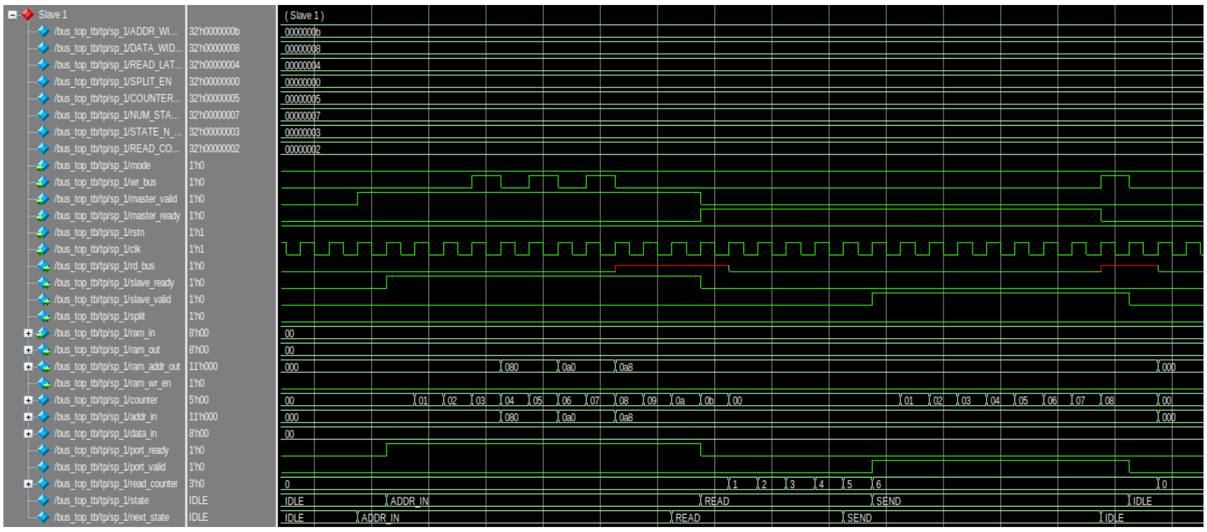
8.2. Master write transaction.



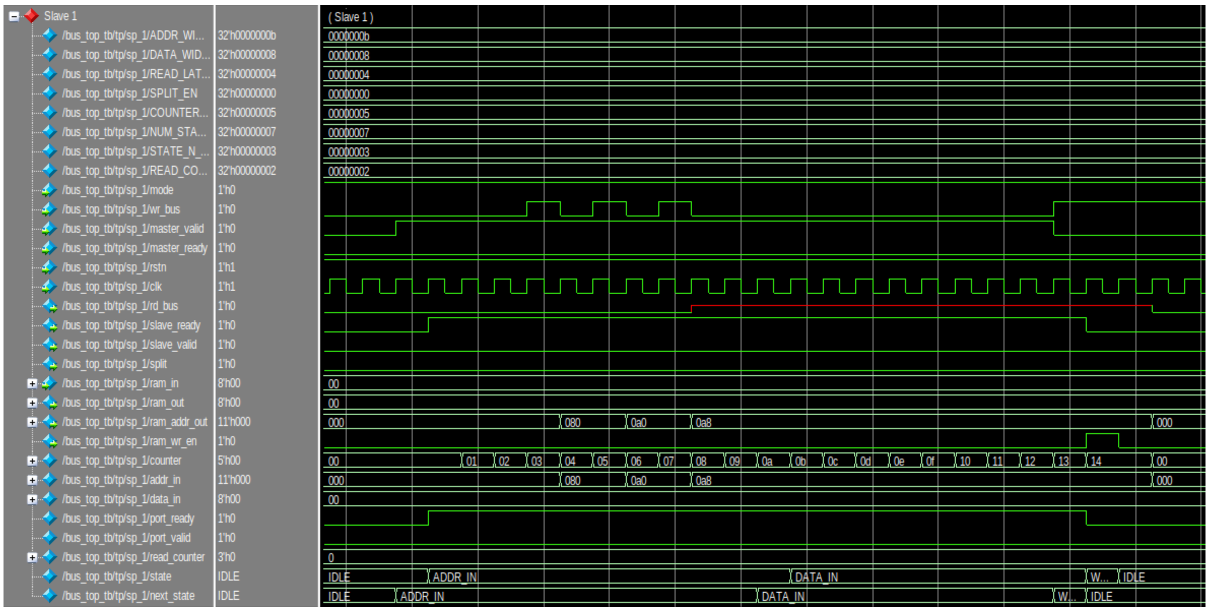
8.3. Master goes split.



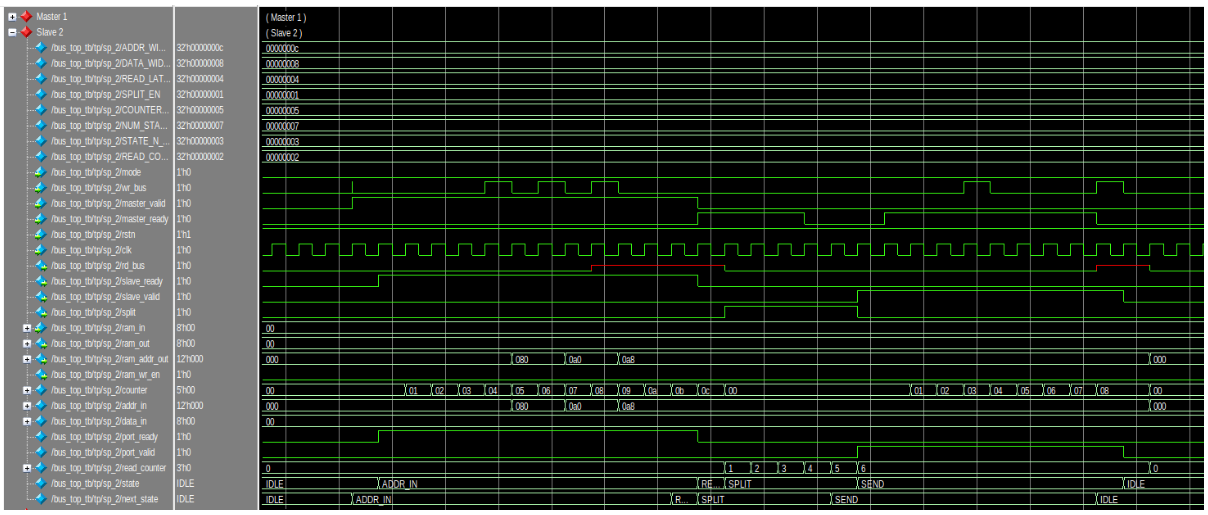
8.4. Slave read transaction.


Slave 1
/bus_top_tb/tp/sp_1/ADDR_WI... 32'h0000000b
/bus_top_tb/tp/sp_1/DATA_WID... 32'h00000008
/bus_top_tb/tp/sp_1/READ_LAT... 32'h00000004
/bus_top_tb/tp/sp_1/SPLIT_EN 32'h00000000
/bus_top_tb/tp/sp_1/COUNTER... 32'h00000005
/bus_top_tb/tp/sp_1/NUM_STA... 32'h00000007
/bus_top_tb/tp/sp_1/STATE_N... 32'h00000003
/bus_top_tb/tp/sp_1/READ_CO... 32'h00000002
/bus_top_tb/tp/sp_1/mode 1'h0
/bus_top_tb/tp/sp_1/wr_bus 1'h0
/bus_top_tb/tp/sp_1/master_valid 1'h0
/bus_top_tb/tp/sp_1/master_ready 1'h0
/bus_top_tb/tp/sp_1/rstn 1'h1
/bus_top_tb/tp/sp_1/clk 1'h1
/bus_top_tb/tp/sp_1/rd_bus 1'h0
/bus_top_tb/tp/sp_1/slave_ready 1'h0
/bus_top_tb/tp/sp_1/slave_valid 1'h0
/bus_top_tb/tp/sp_1/split 1'h0
/bus_top_tb/tp/sp_1/ram_in 8'h00
/bus_top_tb/tp/sp_1/ram_out 8'h00
/bus_top_tb/tp/sp_1/ram_addr_out 11'h000
/bus_top_tb/tp/sp_1/ram_wr_en 1'h0
/bus_top_tb/tp/sp_1/counter 5'h00
/bus_top_tb/tp/sp_1/addr_in 11'h000
/bus_top_tb/tp/sp_1/data_in 8'h00
/bus_top_tb/tp/sp_1/port_ready 1'h0
/bus_top_tb/tp/sp_1/port_valid 1'h0
/bus_top_tb/tp/sp_1/read_counter 3'h0
/bus_top_tb/tp/sp_1/state IDLE
/bus_top_tb/tp/sp_1/next_state IDLE
(Slave 1)
IDLE
ADDR_IN
READ
SEND
IDLE

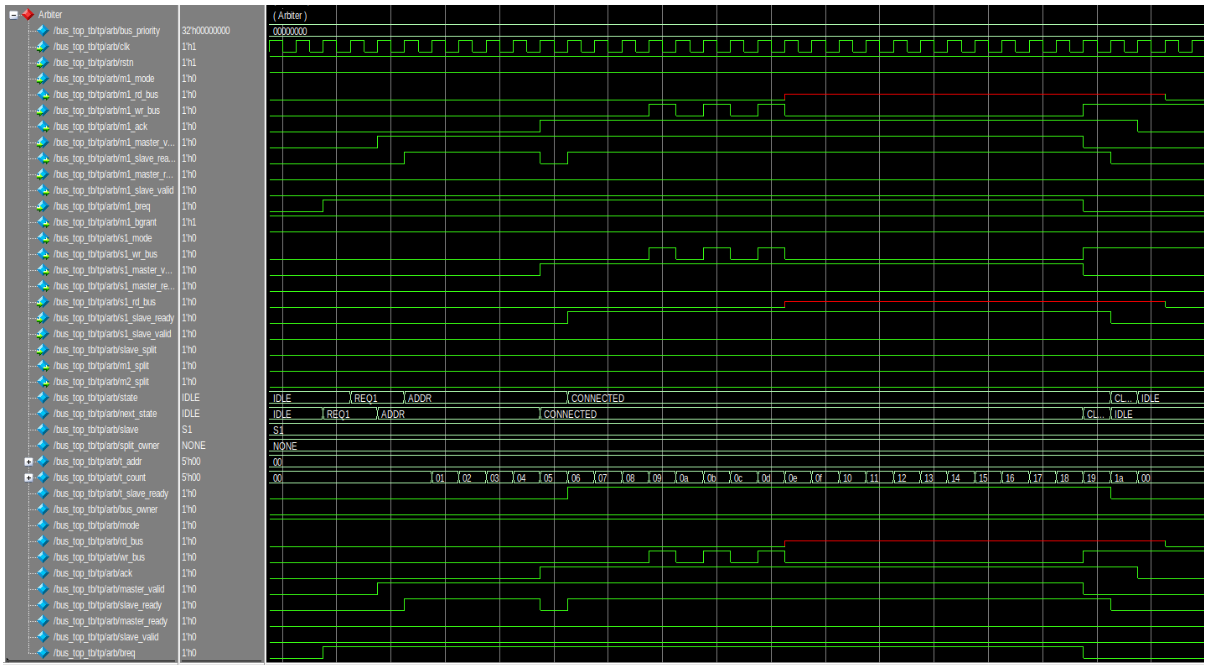
8.5. Slave write transaction.


Slave 1
/bus_top_tb/tp/sp_1/ADDR_WI... 32'h0000000b
/bus_top_tb/tp/sp_1/DATA_WID... 32'h00000008
/bus_top_tb/tp/sp_1/READ_LAT... 32'h00000004
/bus_top_tb/tp/sp_1/SPLIT_EN 32'h00000000
/bus_top_tb/tp/sp_1/COUNTER... 32'h00000005
/bus_top_tb/tp/sp_1/NUM_STA... 32'h00000007
/bus_top_tb/tp/sp_1/STATE_N... 32'h00000003
/bus_top_tb/tp/sp_1/READ_CO... 32'h00000002
/bus_top_tb/tp/sp_1/mode 1'h0
/bus_top_tb/tp/sp_1/wr_bus 1'h0
/bus_top_tb/tp/sp_1/master_valid 1'h0
/bus_top_tb/tp/sp_1/master_ready 1'h0
/bus_top_tb/tp/sp_1/rstn 1'h1
/bus_top_tb/tp/sp_1/clk 1'h1
/bus_top_tb/tp/sp_1/rd_bus 1'h0
/bus_top_tb/tp/sp_1/slave_ready 1'h0
/bus_top_tb/tp/sp_1/slave_valid 1'h0
/bus_top_tb/tp/sp_1/split 1'h0
/bus_top_tb/tp/sp_1/ram_in 8'h00
/bus_top_tb/tp/sp_1/ram_out 8'h00
/bus_top_tb/tp/sp_1/ram_addr_out 11'h000
/bus_top_tb/tp/sp_1/ram_wr_en 1'h0
/bus_top_tb/tp/sp_1/counter 5'h00
/bus_top_tb/tp/sp_1/addr_in 11'h000
/bus_top_tb/tp/sp_1/data_in 8'h00
/bus_top_tb/tp/sp_1/port_ready 1'h0
/bus_top_tb/tp/sp_1/port_valid 1'h0
/bus_top_tb/tp/sp_1/read_counter 3'h0
/bus_top_tb/tp/sp_1/state IDLE
/bus_top_tb/tp/sp_1/next_state IDLE
(Slave 1)
IDLE
ADDR_IN
DATA_IN
W...
IDLE

8.6. Slave split transaction.


Master 1
Slave 2
/bus_top_tb/tp/sp_2/ADDR_WI... 32'h0000000c
/bus_top_tb/tp/sp_2/DATA_WID... 32'h00000008
/bus_top_tb/tp/sp_2/READ_LAT... 32'h00000004
/bus_top_tb/tp/sp_2/SPLIT_EN 32'h00000001
/bus_top_tb/tp/sp_2/COUNTER... 32'h00000005
/bus_top_tb/tp/sp_2/NUM_STA... 32'h00000007
/bus_top_tb/tp/sp_2/STATE_N... 32'h00000003
/bus_top_tb/tp/sp_2/READ_CO... 32'h00000002
/bus_top_tb/tp/sp_2/mode 1'h0
/bus_top_tb/tp/sp_2/wr_bus 1'h0
/bus_top_tb/tp/sp_2/master_valid 1'h0
/bus_top_tb/tp/sp_2/master_ready 1'h0
/bus_top_tb/tp/sp_2/rstn 1'h1
/bus_top_tb/tp/sp_2/clk 1'h0
/bus_top_tb/tp/sp_2/rd_bus 1'h0
/bus_top_tb/tp/sp_2/slave_ready 1'h0
/bus_top_tb/tp/sp_2/slave_valid 1'h0
/bus_top_tb/tp/sp_2/split 1'h0
/bus_top_tb/tp/sp_2/ram_in 8'h00
/bus_top_tb/tp/sp_2/ram_out 8'h00
/bus_top_tb/tp/sp_2/ram_addr_out 12'h000
/bus_top_tb/tp/sp_2/ram_wr_en 1'h0
/bus_top_tb/tp/sp_2/counter 5'h00
/bus_top_tb/tp/sp_2/addr_in 12'h000
/bus_top_tb/tp/sp_2/data_in 8'h00
/bus_top_tb/tp/sp_2/port_ready 1'h0
/bus_top_tb/tp/sp_2/port_valid 1'h0
/bus_top_tb/tp/sp_2/read_counter 3'h0
/bus_top_tb/tp/sp_2/state IDLE
/bus_top_tb/tp/sp_2/next_state IDLE
(Master 1)
(Slave 2)
IDLE
ADDR_IN
RE...
SPLIT
SEND
IDLE

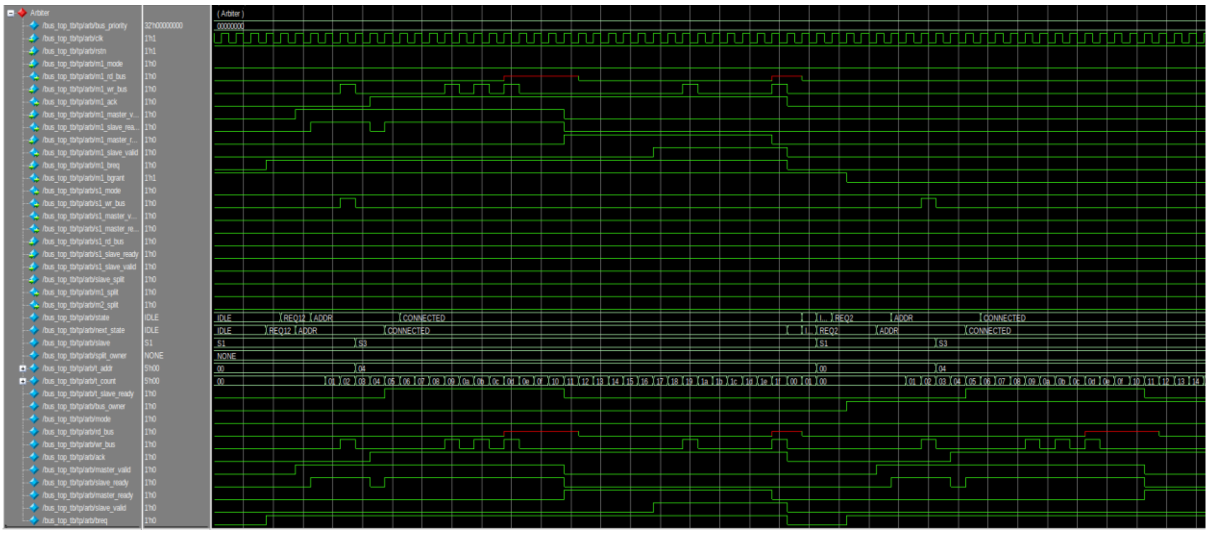
8.7. Arbiter master 1 request transaction.



```
Arbiter
/bus_top_tb/tp/arb/bus_priority      32'h00000000
/bus_top_tb/tp/arb/clk               1'h1
/bus_top_tb/tp/arb/rstn              1'h1
/bus_top_tb/tp/arb/m1_mode           1'h0
/bus_top_tb/tp/arb/m1_rd_bus         1'h0
/bus_top_tb/tp/arb/m1_wr_bus         1'h0
/bus_top_tb/tp/arb/m1_ack            1'h0
/bus_top_tb/tp/arb/m1_master_v...    1'h0
/bus_top_tb/tp/arb/m1_slave_rea...   1'h0
/bus_top_tb/tp/arb/m1_master_r...    1'h0
/bus_top_tb/tp/arb/m1_slave_valid    1'h0
/bus_top_tb/tp/arb/m1_breq           1'h0
/bus_top_tb/tp/arb/m1_bgrant         1'h1
/bus_top_tb/tp/arb/s1_mode           1'h0
/bus_top_tb/tp/arb/s1_wr_bus         1'h0
/bus_top_tb/tp/arb/s1_master_v...    1'h0
/bus_top_tb/tp/arb/s1_master_re...   1'h0
/bus_top_tb/tp/arb/s1_rd_bus         1'h0
/bus_top_tb/tp/arb/s1_slave_ready    1'h0
/bus_top_tb/tp/arb/s1_slave_valid    1'h0
/bus_top_tb/tp/arb/slave_split       1'h0
/bus_top_tb/tp/arb/m1_split          1'h0
/bus_top_tb/tp/arb/m2_split          1'h0
/bus_top_tb/tp/arb/state             IDLE
/bus_top_tb/tp/arb/next_state        IDLE
/bus_top_tb/tp/arb/slave             S1
/bus_top_tb/tp/arb/split_owner       NONE
/bus_top_tb/tp/arb/t_addr            5'h00
/bus_top_tb/tp/arb/t_count           5'h00
/bus_top_tb/tp/arb/t_slave_ready     1'h0
/bus_top_tb/tp/arb/bus_owner         1'h0
/bus_top_tb/tp/arb/mode              1'h0
/bus_top_tb/tp/arb/rd_bus            1'h0
/bus_top_tb/tp/arb/wr_bus            1'h0
/bus_top_tb/tp/arb/ack               1'h0
/bus_top_tb/tp/arb/master_valid      1'h0
/bus_top_tb/tp/arb/slave_ready       1'h0
/bus_top_tb/tp/arb/master_ready      1'h0
/bus_top_tb/tp/arb/slave_valid       1'h0
/bus_top_tb/tp/arb/breq              1'h0
```

(Arbiter)
00000000
IDLE
REQ1
ADDR
CONNECTED
CL...
IDLE
S1
NONE
00
01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17 18 19 1a 00

8.8. Arbiter both master request transaction.



```
Arbiter
/bus_top_tb/tp/arb/bus_priority      32'h00000000
/bus_top_tb/tp/arb/clk               1'h1
/bus_top_tb/tp/arb/rstn              1'h1
/bus_top_tb/tp/arb/m1_mode           1'h0
/bus_top_tb/tp/arb/m1_rd_bus         1'h0
/bus_top_tb/tp/arb/m1_wr_bus         1'h0
/bus_top_tb/tp/arb/m1_ack            1'h0
/bus_top_tb/tp/arb/m1_master_v...    1'h0
/bus_top_tb/tp/arb/m1_slave_rea...   1'h0
/bus_top_tb/tp/arb/m1_master_r...    1'h0
/bus_top_tb/tp/arb/m1_slave_valid    1'h0
/bus_top_tb/tp/arb/m1_breq           1'h0
/bus_top_tb/tp/arb/m1_bgrant         1'h1
/bus_top_tb/tp/arb/s1_mode           1'h0
/bus_top_tb/tp/arb/s1_wr_bus         1'h0
/bus_top_tb/tp/arb/s1_master_v...    1'h0
/bus_top_tb/tp/arb/s1_master_re...   1'h0
/bus_top_tb/tp/arb/s1_rd_bus         1'h0
/bus_top_tb/tp/arb/s1_slave_ready    1'h0
/bus_top_tb/tp/arb/s1_slave_valid    1'h0
/bus_top_tb/tp/arb/slave_split       1'h0
/bus_top_tb/tp/arb/m1_split          1'h0
/bus_top_tb/tp/arb/m2_split          1'h0
/bus_top_tb/tp/arb/state             IDLE
/bus_top_tb/tp/arb/next_state        IDLE
/bus_top_tb/tp/arb/slave             S1
/bus_top_tb/tp/arb/split_owner       NONE
/bus_top_tb/tp/arb/t_addr            5'h00
/bus_top_tb/tp/arb/t_count           5'h00
/bus_top_tb/tp/arb/t_slave_ready     1'h0
/bus_top_tb/tp/arb/bus_owner         1'h0
/bus_top_tb/tp/arb/mode              1'h0
/bus_top_tb/tp/arb/rd_bus            1'h0
/bus_top_tb/tp/arb/wr_bus            1'h0
/bus_top_tb/tp/arb/ack               1'h0
/bus_top_tb/tp/arb/master_valid      1'h0
/bus_top_tb/tp/arb/slave_ready       1'h0
/bus_top_tb/tp/arb/master_ready      1'h0
/bus_top_tb/tp/arb/slave_valid       1'h0
/bus_top_tb/tp/arb/breq              1'h0
```

9. Appendix

9.1. Master

```
module master_port (
    input   logic       clk,
    input   logic       rstn,
    // connections to bus
    output  logic       mode,
    input   logic       rd_bus,
    output  logic       wr_bus,
    input   logic       ack,
    output  logic       master_valid,
    input   logic       slave_ready,
    output  logic       master_ready,
    input   logic       slave_valid,
    output  logic       breq,
    input   logic       bgrant,
    input   logic       split,

    // connections to master
    input   logic[7:0]  m_wr_data,
    output  logic[7:0]  m_rd_data,
    input   logic[15:0] m_addr,
    input   logic       m_mode,
    output  logic       m_wr_en,
    input   logic       m_start
);
    enum logic[3:0] {IDLE, REQ, FETCH, ADDR_1, ADDR_2, WR_DATA, RD_DATA, CLEAN, SPLIT, TIMEOUT_STATE} state, next_state;
    localparam TIMEOUT = 64;

    logic[3:0]  t_count;
    logic[$clog2(TIMEOUT)-1:0] timeout;
    logic[7:0]  t_wr_data;
    logic[7:0]  t_rd_data;
    logic[15:0] t_addr;
    logic       t_mode;

    always_comb begin : NEXT_STATE_LOGIC
        case (state)
            IDLE:               next_state = m_start ? REQ : IDLE;
            REQ:                next_state = bgrant ? FETCH : REQ;
            FETCH:              next_state = ADDR_1;
            ADDR_1:             next_state = timeout != TIMEOUT - 1 ? ((t_count == 5 & slave_ready) ? (ack ? ADDR_2 : CLEAN) : ADDR_1) : TIMEOUT_STATE;
            TIMEOUT_STATE:      next_state = REQ;
            ADDR_2:             next_state = (t_count == 15 & slave_ready) ? (t_mode ? WR_DATA : RD_DATA) : ADDR_2;
            WR_DATA:            next_state = (t_count == 7  & slave_ready) ? IDLE : WR_DATA;
            RD_DATA:            next_state = split == 0 ? ((t_count == 7 & slave_valid) ? CLEAN : RD_DATA) : SPLIT;
            SPLIT:              next_state = split ? SPLIT : RD_DATA;
            CLEAN:              next_state = IDLE;
            default:            next_state = IDLE;
        endcase
    end

    always_ff @(posedge clk or negedge rstn) begin : STATE_SEQUENCER
        state <= !rstn ? IDLE : next_state;
    end

    always_comb begin : OUTPUT_LOGIC
        wr_bus   = state == WR_DATA ? t_wr_data[7] : t_addr[15 - t_count];
        mode     = t_mode;
        master_valid = state == ADDR_2 || state == ADDR_1 || state == WR_DATA;
        master_ready = state == RD_DATA;
```

```systemverilog
        m_rd_data = t_rd_data;
        m_wr_en = state == CLEAN & t_count == 8;
        breq = state != IDLE && state != TIMEOUT_STATE;
    end

    always_ff @(posedge clk or negedge rstn) begin : REG_LOGIC
        if (!rstn) begin
            t_count   <= 0;
            t_wr_data <= 0;
            t_rd_data <= 0;
            t_addr    <= 0;
            t_mode    <= 0;
            timeout   <= 0;
        end else begin
            case (state)
                IDLE: begin
                    t_wr_data <= m_wr_data;
                    t_addr    <= m_addr;
                    t_mode    <= m_mode;
                end

                REQ: begin
                    timeout <= 0;
                    t_count <= 0;
                end

                ADDR_1:begin
                    timeout <= timeout + 1;

                    if(slave_ready) begin
                        t_count <= t_count + 1;
                    end
                end

                ADDR_2: if(slave_ready) begin
                    t_count <= t_count + 1;
                end

                WR_DATA: if(slave_ready) begin
                    t_wr_data <= t_wr_data << 1;
                    t_count <= t_count + 1;
                end

                RD_DATA: if(slave_valid) begin
                    t_rd_data <= {t_rd_data[6:0], rd_bus};
                    t_count <= t_count + 1;
                end

                CLEAN: begin
                    t_count <= 0;
                    t_wr_data <= 0;
                    t_rd_data <= 0;
                    t_addr    <= 0;
                    t_mode    <= 0;
                    timeout   <= 0;
                end
            endcase
        end
    end
endmodule
```

9.2. Slave

```systemverilog
module slave_port_v2#(
    parameter ADDR_WIDTH = 16,
    parameter DATA_WIDTH = 8,
    parameter READ_LATENCY = 4,
    parameter SPLIT_EN = 0
)(
    input logic mode, wr_bus, master_valid, master_ready, rstn, clk,
    output logic rd_bus, slave_ready, slave_valid, split,
    input   logic[DATA_WIDTH-1:0]   ram_in,
    output  logic[DATA_WIDTH-1:0]   ram_out,
    output  logic[ADDR_WIDTH-1:0]   ram_addr_out,
    output  logic                   ram_wr_en
);

// local parameters
localparam COUNTER_LENGTH = $clog2(ADDR_WIDTH+DATA_WIDTH);
localparam NUM_STATES = 7;
localparam STATE_N_BITS = $clog2(NUM_STATES);
localparam READ_COUNTER_LENGTH = $clog2(READ_LATENCY);

// internal signals
logic [COUNTER_LENGTH-1:0]counter;
logic [ADDR_WIDTH-1:0]addr_in;
logic [DATA_WIDTH-1:0]data_in;
logic port_ready, port_valid;
logic [READ_COUNTER_LENGTH:0]read_counter;

// definition of states
enum logic[STATE_N_BITS-1:0] {IDLE, ADDR_IN, DATA_IN, WRITE, READ, SEND, SPLIT} state, next_state;

assign slave_ready = port_ready;
assign slave_valid = port_valid;
assign ram_wr_en = (state == WRITE);
assign ram_addr_out = addr_in;
assign ram_out = data_in;

always_comb begin : NEXT_STATE_DECODER
    case (state)
        IDLE: next_state = ( ( master_valid == 1 ) ? ADDR_IN : IDLE );
        ADDR_IN: next_state = ( (counter < ADDR_WIDTH-1) ? ( (port_ready == 1 && master_valid == 1 ) ? ADDR_IN : IDLE ) : ( (mode == 1) ? DATA_IN : READ ) );
        DATA_IN: next_state = ( (counter < ADDR_WIDTH+DATA_WIDTH) ? ( ( port_ready == 1 && master_valid == 1 ) ? DATA_IN : IDLE ) : WRITE );
        WRITE: next_state = IDLE;
        READ: next_state = ( (read_counter == READ_LATENCY+1) ? SEND : ( (SPLIT_EN ? SPLIT : READ) ) );
        SPLIT: next_state = ( (read_counter == READ_LATENCY+1) ? SEND : SPLIT);
        SEND: next_state = ( (counter < DATA_WIDTH) ? SEND : IDLE );
        default: next_state = IDLE;
    endcase
end

always_ff@(posedge clk or negedge rstn) begin : STATE_SEQUENCER
    if (!rstn) state <= IDLE;
    else state <= next_state;
end

// OUTPUT DECODER
assign port_ready = (state == ADDR_IN) | (state == DATA_IN);
assign port_valid = (state == SEND);
assign split = (state == SPLIT);
assign rd_bus = ram_in[DATA_WIDTH-1-counter];

always_ff@(posedge clk) begin : OUTPUT_DECODER
    case (state)
        IDLE: begin
            counter <= 0;
            addr_in <= 0;
            data_in <= 0;
            read_counter <= 0;
        end

        ADDR_IN: begin
            addr_in[ADDR_WIDTH-1-counter] <= wr_bus;
            counter <= counter + 1;
        end

        DATA_IN: begin
            data_in[DATA_WIDTH-1+ADDR_WIDTH-counter] <= wr_bus;
            counter <= counter + 1;
        end

        READ: begin
            counter <= 0;
            read_counter <= read_counter + 1;
        end

        SPLIT: begin
            read_counter <= read_counter + 1;
        end

        SEND: begin
            if (master_ready == 1) counter <= counter + 1;
        end
    endcase
end
endmodule
```

9.3. Arbiter

```
module arbiter#(
    parameter  bus_priority = 0
)(
    input   logic       clk,
    input   logic       rstn,

    // connections to master 1
    input   logic       m1_mode,
    output  logic       m1_rd_bus,
    input   logic       m1_wr_bus,
    output  logic       m1_ack,
    input   logic       m1_master_valid,
    output  logic       m1_slave_ready,
    input   logic       m1_master_ready,
    output  logic       m1_slave_valid,
    input   logic       m1_breq,
    output  logic       m1_bgrant,

    // connections to master 2
    input   logic       m2_mode,
    output  logic       m2_rd_bus,
    input   logic       m2_wr_bus,
    output  logic       m2_ack,
    input   logic       m2_master_valid,
    output  logic       m2_slave_ready,
    input   logic       m2_master_ready,
    output  logic       m2_slave_valid,
    input   logic       m2_breq,
    output  logic       m2_bgrant,

    // connections to slave 1
    output  logic       s1_mode,
    output  logic       s1_wr_bus,
    output  logic       s1_master_valid,
    output  logic       s1_master_ready,
    input   logic       s1_rd_bus,
    input   logic       s1_slave_ready,
    input   logic       s1_slave_valid,

    // connections to slave 2
    output  logic       s2_mode,
    output  logic       s2_wr_bus,
    output  logic       s2_master_valid,
    output  logic       s2_master_ready,
    input   logic       s2_rd_bus,
    input   logic       s2_slave_ready,
    input   logic       s2_slave_valid,

    // connections to slave 3
    output  logic       s3_mode,
    output  logic       s3_wr_bus,
    output  logic       s3_master_valid,
    output  logic       s3_master_ready,
    input   logic       s3_rd_bus,
    input   logic       s3_slave_ready,
    input   logic       s3_slave_valid,

    // connections to bus bridge
    output  logic       bb_mode,
    output  logic       bb_wr_bus,
    output  logic       bb_master_valid,
    output  logic       bb_master_ready,
    input   logic       bb_rd_bus,
    input   logic       bb_slave_ready,
    input   logic       bb_slave_valid,

    input   logic       slave_split,
    output  logic       m1_split,
    output  logic       m2_split
);

    enum logic[2:0] {IDLE, ADDR, CONNECTED, CLEAN, REQ1, REQ2, REQ12, SPLIT} state, next_state;
    enum logic[1:0] {S1, S2, S3, BB} slave;
    enum logic[1:0] {NONE, M1, M2} split_owner;

    logic[4:0]  t_addr;
    logic[4:0]  t_count;
    logic       t_slave_ready;
    logic       bus_owner;

    logic       mode;
    logic       rd_bus;
    logic       wr_bus;
    logic       ack;
    logic       master_valid;
    logic       slave_ready;
    logic       master_ready;
    logic       slave_valid;
    logic       breq;

    always_comb begin : NEXT_STATE_LOGIC
        case (state)
            IDLE:               begin
                                    if (split_owner == NONE)
                                        next_state = ( (m1_breq == 0) ? ( (m2_breq == 0) ? IDLE : REQ2 ) : ( (m2_breq == 0) ? REQ1 : REQ12 ) );
                                    else if (slave_split == 0)
                                        next_state = CONNECTED;
                                    else
                                        next_state = ( (m1_breq == 0 || split_owner == M1) ? ( (m2_breq == 0 || split_owner == M2) ? IDLE : REQ2 ) : ( (m2_breq == 0 || split_owner == M2) ? REQ1 : REQ12 ) );
                                end
            REQ1:               next_state = ( (m1_breq == 0) ? IDLE : (m1_master_valid) ? ADDR : REQ1 );
            REQ2:               next_state = ( (m2_breq == 0) ? IDLE : (m2_master_valid) ? ADDR : REQ2 );
            REQ12:              next_state = ( (m1_breq == 0 || m2_breq == 0) ? IDLE : (m1_master_valid) ? ADDR : REQ12 );
            ADDR:               next_state = (t_count == 5 && master_valid) ? (ack ? CONNECTED : CLEAN) : ADDR;
            CONNECTED:          next_state = (breq == 0) ? CLEAN : ((slave_split && (t_addr[4:1] == 4'b0001)) ? SPLIT : CONNECTED);
            SPLIT:              next_state = IDLE;
            CLEAN:              next_state = IDLE;
            default:            next_state = IDLE;
        endcase
    end

    always_ff @(posedge clk or negedge rstn) begin : STATE_SEQUENCER
        state <= !rstn ? IDLE : next_state;
    end

    assign slave_ready = ack ? t_slave_ready : state == ADDR;
    assign m1_bgrant = !bus_owner;
    assign m2_bgrant = bus_owner;
    assign m1_split = (split_owner == M1 ? slave_split : 0);
    assign m2_split = (split_owner == M2 ? slave_split : 0);

    always_comb begin
        if (bus_priority == 0) begin
            case (bus_owner)
                1'b1: begin
                    mode           = m2_mode;
                    m2_rd_bus      = rd_bus;
                    wr_bus         = m2_wr_bus;
                    m2_ack         = ack;
                    master_valid   = m2_master_valid;
                    m2_slave_ready = slave_ready;
                    master_ready   = m2_master_ready;
                    m2_slave_valid = slave_valid;
                    breq           = m2_breq;

                    m1_rd_bus      = 0;
                    m1_ack         = 0;
                    m1_slave_ready = 0;
                    m1_slave_valid = 0;
                end

                default: begin
                    mode           = m1_mode;
                    m1_rd_bus      = rd_bus;
                    wr_bus         = m1_wr_bus;
                    m1_ack         = ack;
                    master_valid   = m1_master_valid;
                    m1_slave_ready = slave_ready;
                    master_ready   = m1_master_ready;
                    m1_slave_valid = slave_valid;
                    breq           = m1_breq;

                    m2_rd_bus      = 0;
                    m2_ack         = 0;
                    m2_slave_ready = 0;
                    m2_slave_valid = 0;
                end
            endcase
        end else begin
            case (bus_owner)
                1'b0: begin
                    mode           = m1_mode;
                    m1_rd_bus      = rd_bus;
                    wr_bus         = m1_wr_bus;
                    m1_ack         = ack;
                    master_valid   = m1_master_valid;
                    m1_slave_ready = slave_ready;
                    master_ready   = m1_master_ready;
                    m1_slave_valid = slave_valid;
                    breq           = m1_breq;

                    m2_rd_bus      = 0;
                    m2_ack         = 0;
                    m2_slave_ready = 0;
                    m2_slave_valid = 0;
                end

                default: begin
                    mode           = m2_mode;
                    m2_rd_bus      = rd_bus;
                    wr_bus         = m2_wr_bus;
                    m2_ack         = ack;
                    master_valid   = m2_master_valid;
                    m2_slave_ready = slave_ready;
                    master_ready   = m2_master_ready;
                    m2_slave_valid = slave_valid;
                    breq           = m2_breq;

                    m1_rd_bus      = 0;
                    m1_ack         = 0;
                    m1_slave_ready = 0;
                    m1_slave_valid = 0;
                end
            endcase
        end
    end

    always_comb begin : OUTPUT_LOGIC
        case (t_addr[4:3])
            2'b11: begin
                slave = BB;
                ack = t_count > 1;
            end

            2'b00: begin
                case (t_addr[2:1])
                    2'b01: begin
                        slave = S2;
                        ack = t_count > 3;
                    end
                    2'b10: begin
                        slave = S3;
                        ack = t_count > 3;
                    end
                    2'b00: begin
                        case (t_addr[0])
                            1'b0: begin
                                slave = S1;
                                ack = t_count > 4;
                            end
                            1'b1: begin
                                slave = S1;
                                ack = 0;
                            end
                        endcase
                    end
                    default: begin
                        slave = S1;
                        ack = 0;
                    end
                endcase
            end
            default: begin
                slave = S1;
                ack = 0;
            end
        endcase

        case (slave)
```

```
            S1: begin
                s1_mode         = mode;
                s1_wr_bus       = wr_bus;
                s1_master_valid = master_valid && ack;
                s1_master_ready = master_ready;
                t_slave_ready   = s1_slave_ready;
                slave_valid     = s1_slave_valid;
                rd_bus          = s1_rd_bus;

                s2_mode         = 0;
                s2_wr_bus       = 0;
                s2_master_valid = 0;
                s2_master_ready = 0;

                s3_mode         = 0;
                s3_wr_bus       = 0;
                s3_master_valid = 0;
                s3_master_ready = 0;

                bb_mode         = 0;
                bb_wr_bus       = 0;
                bb_master_valid = 0;
                bb_master_ready = 0;
            end
            S2: begin
                s1_mode         = 0;
                s1_wr_bus       = 0;
                s1_master_valid = 0;
                s1_master_ready = 0;

                s2_mode         = mode;
                s2_wr_bus       = wr_bus;
                s2_master_valid = master_valid && ack;
                s2_master_ready = master_ready;
                t_slave_ready  = s2_slave_ready;
                slave_valid  = s2_slave_valid;
                rd_bus       = s2_rd_bus;

                s3_mode         = 0;
                s3_wr_bus       = 0;
                s3_master_valid = 0;
                s3_master_ready = 0;

                bb_mode         = 0;
                bb_wr_bus       = 0;
                bb_master_valid = 0;
                bb_master_ready = 0;
            end
            S3: begin
                s1_mode         = 0;
                s1_wr_bus       = 0;
                s1_master_valid = 0;
                s1_master_ready = 0;

                s2_mode         = 0;
                s2_wr_bus       = 0;
                s2_master_valid = 0;
                s2_master_ready = 0;

                s3_mode         = mode;
                s3_wr_bus       = wr_bus;
                s3_master_valid = master_valid && ack;
                s3_master_ready = master_ready;
                t_slave_ready  = s3_slave_ready;
                slave_valid  = s3_slave_valid;
                rd_bus       = s3_rd_bus;

                bb_mode         = 0;
                bb_wr_bus       = 0;
                bb_master_valid = 0;
                bb_master_ready = 0;
            end
            BB: begin
                s1_mode         = 0;
                s1_wr_bus       = 0;
                s1_master_valid = 0;
                s1_master_ready = 0;

                s2_mode         = 0;
                s2_wr_bus       = 0;
                s2_master_valid = 0;
                s2_master_ready = 0;

                s3_mode         = 0;
                s3_wr_bus       = 0;
                s3_master_valid = 0;
                s3_master_ready = 0;

                bb_mode         = mode;
                bb_wr_bus       = wr_bus;
                bb_master_valid = master_valid && ack;
                bb_master_ready = master_ready;
                t_slave_ready  = bb_slave_ready;
                slave_valid  = bb_slave_valid;
                rd_bus       = bb_rd_bus;
            end
        endcase
    end

    always_ff @(posedge clk or negedge rstn) begin : REG_LOGIC
        if (!rstn) begin
            t_count     <= 0;
            t_addr      <= 0;
            bus_owner   <= 0;
            split_owner <= NONE;
        end else begin
            case (state)
                IDLE: begin
                    if (split_owner != NONE && slave_split == 0) begin
                        bus_owner <= (split_owner == M1 ? 0 : 1);
                        t_addr[4:1] <= 4'b0001;
                    end
                end

                REQ1: begin
                    bus_owner <= 0;
                end

                REQ2: begin
                    bus_owner <= 1;
                end

                REQ12: begin
                    bus_owner <= bus_priority;
                end

                ADDR: if (master_valid) begin
                    t_count <= t_count + 1;
                    t_addr[4 - t_count] <= wr_bus;
                end

                CONNECTED: begin
                    t_count <= t_count + 1;
                end

                SPLIT: begin
                    split_owner <= (bus_owner == 0) ? M1 : M2;
                end

                CLEAN: begin
```

```
                    t_count <= 0;
                    t_addr  <= 0;

                    if (bus_owner == 0 && split_owner == M1) split_owner <= NONE;
                    else if (bus_owner == 1 && split_owner == M2) split_owner <= NONE;
                end
            endcase
        end
    end
endmodule
```

9.4. Master Bus Bridge

```
module bb_master_port (
    input   logic       clk,
    input   logic       rstn,
    // connections to bus
    output  logic       mode,
    input   logic       rd_bus,
    output  logic       wr_bus,
    input   logic       ack,
    output  logic       master_valid,
    input   logic       slave_ready,
    output  logic       master_ready,
    input   logic       slave_valid,
    output  logic       breq,
    input   logic       bgrant,
    input   logic       split,

    // connections to master
    input   logic[24:0] fifo_data_in,
    output  logic[7:0]  uart_register_out,
    output  logic       m_out_valid,
    input   logic       fifo_empty,
    output  logic       fifo_deq
);
    enum logic[3:0] {IDLE, REQ, ADDR_1, ADDR_2, WR_DATA, RD_DATA, CLEAN, SPLIT, TIMEOUT_STATE, FETCH, DEQ} state, next_state;
    localparam TIMEOUT = 64;

    logic[3:0]  t_count;
    logic[$clog2(TIMEOUT)-1:0] timeout;
    logic[7:0]  t_wr_data;
    logic[7:0]  t_rd_data;
    logic[15:0] t_addr;
    logic       t_mode;

    always_comb begin : NEXT_STATE_LOGIC
        case (state)
            IDLE:               next_state = !fifo_empty ? DEQ : IDLE;
            DEQ:                next_state = FETCH;
            FETCH:              next_state = REQ;
            REQ:                next_state = bgrant ? ADDR_1 : REQ;
            ADDR_1:             next_state = timeout != TIMEOUT - 1 ? ((t_count == 5 & slave_ready) ? (ack ? ADDR_2 : CLEAN) : ADDR_1) : TIMEOUT_STATE;
            TIMEOUT_STATE:      next_state = REQ;
            ADDR_2:             next_state = (t_count == 15 & slave_ready) ? (t_mode ? WR_DATA : RD_DATA) : ADDR_2;
            WR_DATA:            next_state = (t_count == 7  & slave_ready) ? IDLE : WR_DATA;
            RD_DATA:            next_state = split == 0 ? ((t_count == 7 & slave_valid) ? CLEAN : RD_DATA) : SPLIT;
            SPLIT:              next_state = split ? SPLIT : RD_DATA;
            CLEAN:              next_state = IDLE;
            default:            next_state = IDLE;
        endcase
    end

    always_ff @(posedge clk or negedge rstn) begin : STATE_SEQUENCER
        state <= !rstn ? IDLE : next_state;
    end

    always_comb begin : OUTPUT_LOGIC
        wr_bus  = state == WR_DATA ? t_wr_data[7] : t_addr[15 - t_count];
        mode    = t_mode;
        master_valid = state == ADDR_2 || state == ADDR_1 || state == WR_DATA;
        master_ready = state == RD_DATA;
        uart_register_out = t_rd_data;
        m_out_valid = state == CLEAN & t_count == 8;
        breq = state != IDLE && state != TIMEOUT_STATE;
        fifo_deq = state == DEQ;
    end

    always_ff @(posedge clk or negedge rstn) begin : REG_LOGIC
        if (!rstn) begin
            t_count   <= 0;
            t_wr_data <= 0;
            t_rd_data <= 0;
            t_addr    <= 0;
            t_mode    <= 0;
            timeout   <= 0;
        end else begin
            case (state)
                FETCH: begin
                    t_wr_data <= fifo_data_in[7:0];
                    t_addr    <= fifo_data_in[23:8];
                    t_mode    <= fifo_data_in[24];
                end

                REQ: begin
                    timeout <= 0;
                    t_count <= 0;
                end

                ADDR_1:begin
                    timeout <= timeout + 1;

                    if(slave_ready) begin
                        t_count <= t_count + 1;
                    end
                end

                ADDR_2: if(slave_ready) begin
                    t_count <= t_count + 1;
                end

                WR_DATA: if(slave_ready) begin
                    t_wr_data <= t_wr_data << 1;
                    t_count <= t_count + 1;
                end

                RD_DATA: if(slave_valid) begin
                    t_rd_data <= {t_rd_data[6:0], rd_bus};
                    t_count <= t_count + 1;
                end

                CLEAN: begin
                    t_count <= 0;
                    t_wr_data <= 0;
                    t_rd_data <= 0;
                    t_addr    <= 0;
                    t_mode    <= 0;
                    timeout   <= 0;
                end
            endcase
        end
    end
endmodule
```

9.5. Slave Bus Bridge

```
module slave_bus_bridge#(
    parameter ADDR_WIDTH = 16,
    parameter DATA_WIDTH = 8,
    parameter SPLIT_EN = 0
)(
    input logic mode, wr_bus, master_valid, master_ready, rstn, clk, valid_in,
    input logic [DATA_WIDTH-1:0]uart_register_in,
    output logic [1+16+DATA_WIDTH-1:0]uart_register_out,
    output logic rd_bus, slave_ready, slave_valid, split, valid_out
);

// local parameters
localparam COUNTER_LENGTH = $clog2(ADDR_WIDTH+DATA_WIDTH);
localparam NUM_STATES = 8;
localparam STATE_N_BITS = $clog2(NUM_STATES);

// internal signals
logic [COUNTER_LENGTH-1:0]counter;
logic [ADDR_WIDTH-1:0]addr_in;
logic [DATA_WIDTH-1:0]data_in;
logic port_ready, port_valid;

// definition of states
enum logic[STATE_N_BITS-1:0] {IDLE, ADDR_IN, DATA_IN, WRITE, READ, SEND, SPLIT, SEND_RD_ADDR} state, next_state;

assign slave_ready = port_ready;
assign slave_valid = port_valid;

always_comb begin : NEXT_STATE_DECODER
    case (state)
        IDLE: next_state = ( ( master_valid == 1 ) ? ADDR_IN : IDLE );
        ADDR_IN: next_state = ( (counter < ADDR_WIDTH-1) ? ( (port_ready == 1 && master_valid == 1 ) ? ADDR_IN : IDLE ) : ( (mode == 1) ? DATA_IN : SEND_RD_ADDR ) );
        DATA_IN: next_state = ( (counter < ADDR_WIDTH+DATA_WIDTH) ? ( ( port_ready == 1 && master_valid == 1 ) ? DATA_IN : IDLE ) : WRITE );
        WRITE: next_state = IDLE;
        SEND_RD_ADDR: next_state = READ;
        READ: next_state = ( (valid_in) ? SEND : ( (SPLIT_EN ? SPLIT : READ) ) );
        SPLIT: next_state = ( (valid_in) ? SEND : SPLIT);
        SEND: next_state = ( (counter < DATA_WIDTH) ? SEND : IDLE );
        default: next_state = IDLE;
    endcase
end

always_ff@(posedge clk or negedge rstn) begin : STATE_SEQUENCER
    if (!rstn) state <= IDLE;
    else state <= next_state;
end

// OUTPUT DECODER
assign port_ready = (state == ADDR_IN) | (state == DATA_IN);
assign port_valid = (state == SEND);
assign split = (state == SPLIT);
assign rd_bus = uart_register_in[DATA_WIDTH-1-counter];

always_ff@(posedge clk) begin : OUTPUT_DECODER
    case (state)
        IDLE: begin
```

```
            counter <= 0;
            addr_in <= 0;
            data_in <= 0;
            valid_out <= 0;
        end

        ADDR_IN: begin
            addr_in[ADDR_WIDTH-1-counter] <= wr_bus;
            counter <= counter + 1;
        end

        DATA_IN: begin
            data_in[DATA_WIDTH-1+ADDR_WIDTH-counter] <= wr_bus;
            counter <= counter + 1;
        end

        WRITE: begin
            uart_register_out <= {mode, 2'b00, addr_in, data_in};
            valid_out <= 1;
        end

        SEND_RD_ADDR: begin
            uart_register_out <= {mode, 2'b00, addr_in, 8'd0};
            valid_out <= 1;
        end

        READ: begin
            valid_out <= 0;
            counter <= 0;
        end

        SEND: begin
            if (master_ready == 1) counter <= counter + 1;
        end
    endcase
end
endmodule
```

9.6. FIFO

```
module FIFO #(parameter WIDTH = 32, parameter DEPTH = 16) (
    input logic clk,
    input logic rstn,
    input logic [WIDTH-1:0] data_in,
    input logic enq,
    input logic deq,
    output logic [WIDTH-1:0] data_out,
    output logic empty
);

logic [DEPTH-1:0][WIDTH-1:0] memory; //packed array
logic [$clog2(DEPTH):0] rd_ptr, wr_ptr;
logic full;

assign data_out = memory[rd_ptr];
assign empty = (rd_ptr == wr_ptr);
assign full = (rd_ptr == wr_ptr + 1);

always_ff @(posedge clk or negedge rstn) begin
    if(!rstn) begin
        rd_ptr <= 0;
        wr_ptr <= 0;
    end

    else begin
```

```
        if (enq && !full) begin
            memory[wr_ptr] <= data_in;
            wr_ptr <= wr_ptr + 1;
        end

        if (deq && !empty) begin
            rd_ptr <= rd_ptr + 1;
        end
    end
end

endmodule
```