AN 585: Simulation Debugging Using Triple Speed Ethernet Testbench

© August 2009 AN-585-1.0

Introduction

This application note shows how you can leverage the verification environment in the testbench provided in the Altera® Triple Speed Ethernet MegaCore® function to debug your system design. You can use the different types of loopback in the testbench to simulate your system design, and create various common scenarios by configuring the parameters and the state machine in the testbench.

The Triple Speed Ethernet MegaCore function consists of a 10/100/1000 Mbps Ethernet media access controller (MAC), a 1000BASE-X physical coding sub-layer (PCS), and an optional physical medium attachment (PMA). The Triple Speed Ethernet MegaCore function supports seamless interface to commercial Ethernet PHY devices via medium independent interface (MII) and gigabit medium independent interface (GMII). The MegaCore function also supports reduced gigabit medium independent interface (RGMII) in 10/100/1000 Mbps.

The Triple Speed Ethernet MegaCore function provides a testbench that supports simulation of all basic Ethernet packet transactions, and has an easy-to-use simulation environment for any standard HDL simulator. The testbench consists of device under test (DUT) modules which are the custom MegaCore function variations, the Ethernet frame generators, and clock and reset generators.

The testbench is intended for simulating common configurations and may not cover all the possible configurations of the Triple Speed Ethernet MegaCore function.

For more information about the Triple Speed Ethernet MegaCore function, refer to the Triple Speed Ethernet MegaCore Function page of the Altera website.

Types of Loopback

You can use the following types of loopback in the testbench to debug your system design:

- No loopback—you can disable loopback through the testbench settings.
- MAC local loopback—you can enable the MAC local loopback through the MegaWizard™ interface.
- PHY loopback—you can enable the PHY loopback through the testbench settings.
- External loopback—you can enable the external loopback through the testbench settings.

If you turn on the **Enable MII/GMII/RGMII loopback logic** option in the MegaWizard interface, the testbench by default configures the Triple Speed Ethernet core to enable the MAC local loopback. You must ensure that the `ENABLE_GMII_LOOPBACK` parameter in the testbench is set to `0` when you set the testbench to operate in the external loopback or PHY loopback.

The ENABLE_GMII_LOOPBACK parameter is listed under the Core settings in the testbench file. With the exception of the ENABLE_GMII_LOOPBACK parameter, the values of the other parameters in the Core settings list must not be changed.

Table 1 shows which loopback is available for the different DUT or variations of the Triple Ethernet MegaCore function.

Table 1. Debugging Modes Support for Variations of Triple Speed Ethernet MegaCore Function

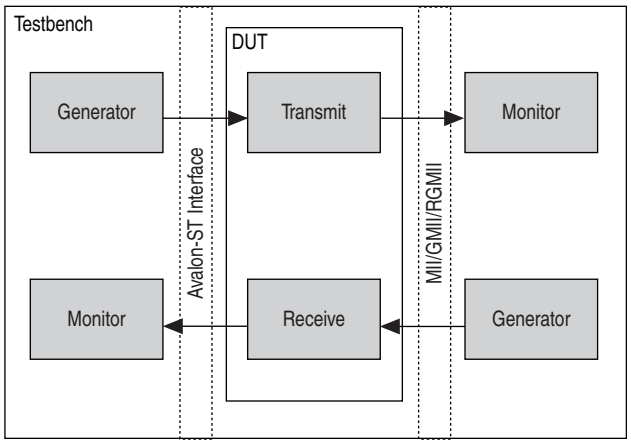
Core Variation	Internal (MAC) Local Loopback	External Loopback	PHY Loopback (GXB)	No Loopback
MAC only	Yes	Yes	No	Yes
MAC and PCS	Yes	Yes	No	No
MAC, PCS and PMA	Yes	Yes	Yes	No
PCS only	No	Yes	No	No
PCS and PMA	No	Yes	Yes	No

The following sections describe the types of loopback in detail.

No Loopback

Figure 1 shows the block diagram of the testbench when loopback is disabled.

Figure 1. Block Diagram of Triple Speed Ethernet Testbench with No Loopback

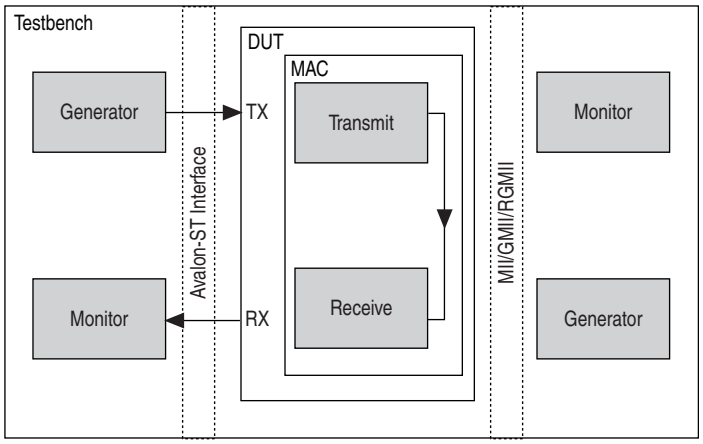


To operate the testbench without a loopback, you must set the TB_RXFRAMES parameter to a value higher than zero in the MAC only core variation.

MAC Local Loopback

Figure 2 shows the block diagram of the testbench when the MAC local loopback is enabled.

Figure 2. Triple Speed Ethernet Testbench with MAC Local Loopback Enabled



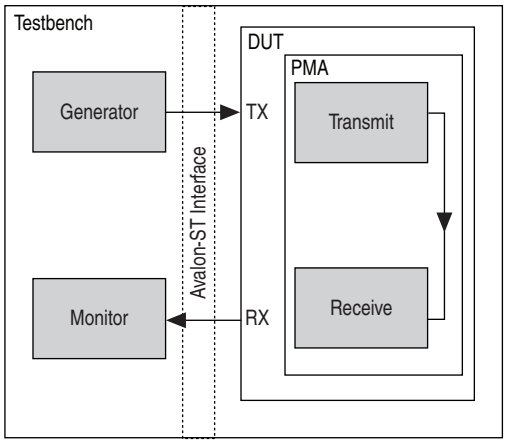
By enabling the MII/GMII/RGMII loopback logic, you set the testbench parameter `ENABLE_GMII_LOOPBACK` to `1`, which in turn sets the `LOOP_ENA` bit to 1 in the `command_config` register.

PHY Loopback

You can enable the PHY loopback in designs that have GX transceivers as PMA modules.

Figure 3 shows the block diagram of the testbench when the PHY loopback is enabled.

Figure 3. Triple Speed Ethernet Testbench in PHY Loopback

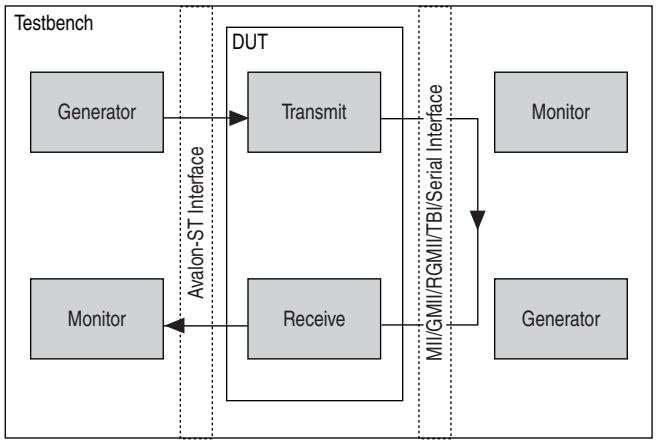


To enable the PHY loopback, you set the `sd_loopback` bit in the PCS control register to `1`. To set the `sd_loopback` bit, you need to do a configuration write to the PCS control register.

External Loopback

Figure 4 shows the block diagram of the testbench when the external loopback is enabled.

Figure 4. Triple Speed Ethernet Testbench in External Loopback



You can enable the external loopback in all core variations, on the following three interfaces:

- Serial interface—For core variations that include PMA, the external loopback is implemented on the serial interface. When a core variation that includes PMA module is selected, the loopback on the serial interface is generated in the testbench by default. You do not need to make any changes to the configuration.
- Ten-bit interface (TBI)—The transceiver data is looped back to the receiver on the TBI. The TBI applies to PCS only, and MAC and PCS core variations.
- MII/GMII/RGMII—The loopback on the MII/GMII/RGMII applies to MAC only core variations. The `TB_RXFRAMES` parameter must be set to `0` to activate the external loopback on the MII/GMII/RGMII.

Customizing a Test Case

You can use the testbench to accelerate the debugging process by duplicating test cases with problems. The testbench, by default, is configured with the following features:

- Gigabit mode enabled (`ETH_MODE = 1000`)
- Loopback mode (`TB_RXFRAMES = 0`)
- The MAC function transmits five normal Ethernet frames (`TB_TXFRAMES = 5`)
- First transmit packet with payload length of 100 bytes (`TB_LENSTART = 100`)
- Increment of one byte in payload length for every subsequent frame (`TB_LENSTEP = 1`)
- Maximum payload length of 1500 bytes (`TB_LENMAX = 1500`)
- Inter packet frame of 12 clocks (`TB_IPG_LENGTH = 12`)

In addition to the default test case, you can create your own customized test cases by simply configuring the testbench parameters, or the VHDL or Verilog HDL codes in the testbench.

Configuring Parameters

You can use the functionality configuration parameters to enable or disable specific functionality of MAC and PCS. You can use the test configuration parameters to create custom test scenarios.

Table 2 shows how you can configure certain parameters to carry out specific tasks.

Table 2. Manipulating Parameters for Specific Tasks (Part 1 of 2)

Task	Parameter	Description
Changing Ethernet speed	ETH_MODE	You can configure the Ethernet speed using the ETH_MODE parameter. The valid values for this parameter are 10, 100, and 1000. The value of the ETH_MODE parameter directly affects the value of the ETH_SPEED and ENA_10 bits in the command_config register. The Triple Speed Ethernet testbench sets these 2 bits accordingly with respect to the value of the ETH_MODE parameter. When the ETH_MODE parameter is set to 10 or 100, the MII is enabled and the Ethernet speed is set to 10 Mbps and 100 Mbps respectively. With the value of 1000, the GMII is enabled and the Ethernet speed is at 1000 Mbps.
Varying frame length	TB_LENSTART, TB_LENSTEP, TB_LENMAX	To modify the frame length of the Ethernet packets, you can configure the TB_LENSTART, TB_LENSTEP, and TB_LENMAX parameters. The TB_LENSTART parameter defines payload length in bytes for the first frame. The subsequent frames have payload lengths of (previous_payload_length + TB_LENSTEP). The TB_LENMAX parameter defines the maximum payload length. When the payload length hits the maximum, the payload length rolls back and starts to increment from 0.
Generating different frame types	TB_ENA_VLAN, TB_TRIGGERXOFF, TB_TRIGGERXON	By default, the frames generated are normal ethernet packets. The TB_ENA_VLAN, TB_TRIGGERXOFF, and TB_TRIGGERXON parameters trigger the generation of the virtual local area network (VLAN) frames and pause frames. You can generate the VLAN frame by setting a non-zero value to the TB_ENA_VLAN parameter. You can generate the pause frames by setting a non-zero value to the TB_TRIGGERXOFF or TB_TRIGGERXON parameters. The TB_TRIGGERXOFF parameter triggers the generation of pause frame with non-zero pause quanta, while the TB_TRIGGERXON parameter triggers the generation of pause frame with zero pause quanta.

Table 2. Manipulating Parameters for Specific Tasks (Part 2 of 2)

Task	Parameter	Description
Controlling the streaming of frames	TB_TXFRAME, TB_RXFRAME, TB_IPG_LENGTH	The number of frames generated by the generator is controlled by the TB_TXFRAME and TB_RXFRAMES parameters. You can control the interpacket gap by varying the TB_IPG_LENGTH parameter.
Inducing collision in half-duplex mode	RX_COL_FRM, RX_COL_GEN, TX_COL_FRM, TX_COL_GEN, TX_COL_NUM	The collision test case is only applicable when half-duplex mode is enabled (HD_ENA = 1). You can configure these parameters to specify the location of the intended collision.

For more information on the testbench simulation parameters, refer to Appendix B, Simulation Parameters in the *Triple Speed Ethernet MegaCore Function User Guide*.

Modifying VHDL or Verilog HDL Code

The following sections describe how to configure the Ethernet frame generator, and change the state machine in the testbench by modifying the VHDL or Verilog HDL code.

Configuring the Ethernet Frame Generator

The Ethernet frame generator is a bus functional model that constructs Ethernet frames to be sent to and from the MAC function. The frame generation is controlled by input parameters in the Ethernet frame generator. For the Ethernet frame generator on the Avalon® Streaming (Avalon-ST) interface, the parameters are prefixed with ff_, and for the Ethernet frame generator on the MII/GMII/RGMII, the parameters are prefixed with gm_. You can modify the input parameters in Table 3 to generate different test cases by substituting <*intf*> with ff_ and gm_ for the respective Ethernet frame generators.

Table 3. Input Parameters (Part 1 of 2)

Input Parameter	Description
<*intf*>_mac_reverse	When enabled, the destination address and source address are sent to the most significant byte (MSB) first.
<*intf*>_dst	Hexadecimal value for the destination address field.
<*intf*>_src	Hexadecimal value for the source address field.
<*intf*>_prmble_len	Number of preamble bytes to be generated.
<*intf*>_pquant	Pause quanta value.
<*intf*>_vlan_ctl	Two bytes, VLAN information for the VLAN tagged frame.
<*intf*>_len	Payload length.
<*intf*>_frmtype	Two bytes, when non null, this value is inserted in the type or frame field instead of the payload length.
<*intf*>_cntstart	Decimal value. Payload length of the first frame.
<*intf*>_cntstep	Decimal value. Number of bytes to increment on subsequent frames.
<*intf*>_ipg_len	Inter-packet gap in decimals.
<*intf*>_payload_err	When set to 1, induces data corruption in payload by corrupting last byte of data.

Table 3. Input Parameters (Part 2 of 2)

Input Parameter	Description
<*intf*>_prmbl_err	When set to 1, induces data corruption in preamble bytes.
<*intf*>_crc_err	When set to 1, inserts incorrect Cyclic Redundancy Check (CRC) to frame.
<*intf*>_vlan_en	When set to 1, VLAN tagged frame is generated.
<*intf*>_stack_vlan_en	When set to 1, stacked VLAN tagged frame is generated.
<*intf*>_pad_en	When set to 1, zero padding to frame is enabled.
<*intf*>_phy_err	When set to 1, asserts the rx_err signal.
<*intf*>_end_err	When set to 1, the rx_dv signal only deasserts one clock cycle after end of frame.
<*intf*>_data_only	When set to 1, omits preamble bytes, zero paddings, and CRC.
<*intf*>_pause_gen	When set to 1, generates a pause frame. This parameter is always set to 1 for the Ethernet frame generator on the Avalon-ST interface. The MAC function generates pause frame for its transmit path.
<*intf*>_carrier_sense *(1)*	When set to 1, simulates the carrier sense.
<*intf*>_false_carrier *(1)*	When set to 1, simulates the false carrier.
<*intf*>_carrier_extend *(1)*	When set to 1, simulates the carrier extension.
<*intf*>_carrier_extend_error *(1)*	When set to 1, simulates the carrier extension with error.

Note to Table 3:

(1) This parameter is only applicable for the Ethernet frame generator on the MII/GMII/RGMII.

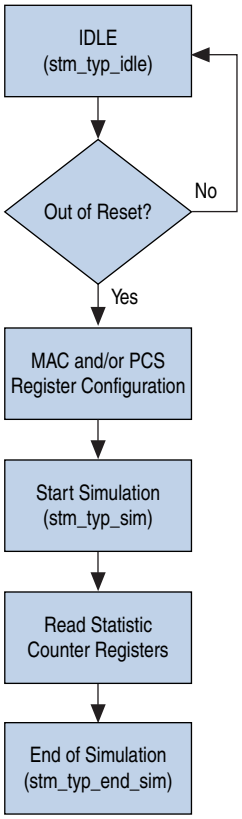
In the testbench, the parameters in Table 3 have default values assigned to them. To change the values, look for the strings that begin with `assign` <*parameter name*> in the Ethernet frame generator configuration settings and modify the parameters to suit your test case requirements.

Changing the State Machine

The state machine in the testbench controls the sequence of the DUT register configuration and the simulation flow. The state machine also implements the control interface signals; mainly the `read`, `write`, and `waitrequest` signals that control the read and write of the MAC and PCS registers.

Figure 5 shows the simulation flow of the Triple Speed Ethernet testbench.

Figure 5. Testbench Simulation Flow



The sequence of the DUT register configuration is fixed in the testbench. To change the sequence, you must change the states prefixed with `stm_` in the state machine.

At each state, a value is assigned to `reg_data_in` and updated in the MAC register, for example:

```
reg_data_in = # (2) 32'h 00000000;
```

Change the value on this assignment so that the value is written to the targeted register.

Test Case Samples

The following section describes test cases that demonstrate how to use the testbench to configure parameters, the Ethernet frame generator, and the state machines.

You can obtain these test cases from the **AN585_test_case.zip** file from the Literature: Application Notes page of the Altera website. Download and unzip the **AN585_test_case.zip** file to the <*your project*> folder. Run the simulation for the test cases by executing the corresponding **.tcl** files from the <*your project*>**\tse_debug_with_tb\testbench\tse_debug_with_tb** directory.

☞ You can compare the testbench files of these test cases (**tb_testcase**<*number*>**.v**) with the default testbench file (**tb_default.v**) to find out the changes made.

Test Case 1

In this test case, the MAC function is operating at the speed of 1000 Mbps. The MAC function transmits four normal packets; with the first packet starting with a payload length of 110 bytes, the second with 120 bytes, the third with 130 bytes, and the fourth with 140 bytes. While the MAC function transmits its second packet, the xoff signal is asserted to generate a pause frame with pause quanta of 8 (4,096 ns).

To reproduce this test case, set the following parameter values:

```
ENABLE_GMII_LOOPBACK = 0;
TB_TXFRAMES = 4;
TB_LENSTART = 110;
TB_LENSTEP = 10;
TB_TRIGGEROFF = 300;
TB_MACPAUSEQ = 8
```

When you run the simulation, the MAC function is operating at the speed of 1000 Mbps and in duplex mode. You can observe data on the gm_tx_d and gm_rx_d signals as the testbench is set to do an external loopback on the GMII. The MAC function transmits four frames: the first with a payload length of 110 bytes, the second with 120 bytes, the third with 130 bytes, and the fourth with 140 bytes. The xoff_gen signal is asserted at 9,528 ns during the transmission of the second frame. The pause frame is sent out by the MAC function as soon as the transmission of the second frame is complete. After the transmission of the third frame, the transmission pauses for 4,096 ns before transmitting the fourth frame. The pause in the transmission indicates the MAC function's response to the pause frame which is being looped back on its receive path.

Test Case 2

In this test case, the MAC function is configured to half-duplex mode and operates at 100 Mbps. The MAC function transmits two frames and receives three frames. A collision occurs on the first frame.

To reproduce this test case, set the following parameter values:

```
ETH_MODE = 100;
HD_ENA = 1'b 1;
TB_RXFRAMES = 3;
TB_TXFRAMES = 2;
TX_COL_FRM = 1;
TX_COL_GEN = 100
```

When you run the simulation, the MAC function is operating at the speed of 100 Mbps, and in half-duplex mode. You can observe data on the m_tx_d and m_rx_d signals on the MII. As the MAC function operates in half-duplex mode, the transaction is only one way at any given time. The MAC function only starts transmitting the frames after its receive operation is complete. A collision is induced on the first frame. Once the collision takes place, the MAC function stops its transmission and sends out a 32-bit jam pattern. After an interval as long as the backoff period, the MAC function retransmits the first packet.

Test Case 3

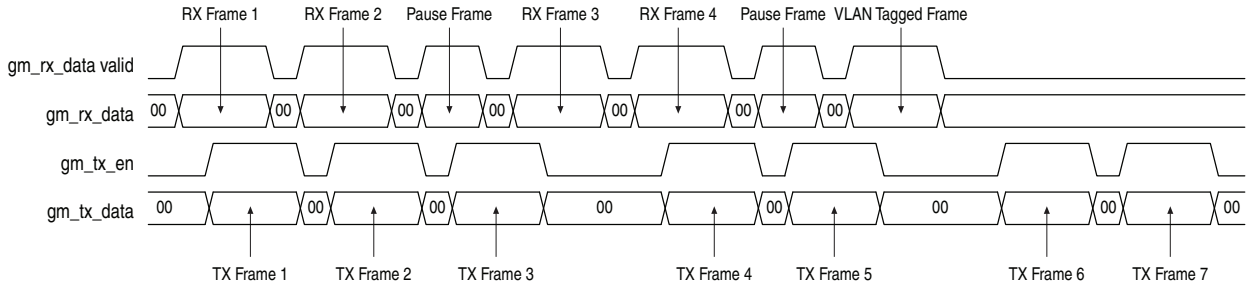
In this test case, the MAC function operates at 1000 Mbps. The destination address of the transmit frame is `0xabcdef221100`. The MAC function transmits seven Ethernet packets while the payload length is arbitrary. On the MAC receive path, the MAC function receives four normal Ethernet frames, two pause frames, and a VLAN tagged frame. Frame 3 and Frame 6 are pause frames, Frame 7 is a VLAN tagged frame, and the rest are normal frames.

To reproduce this test case, set the following parameter values:

```
TB_RXFRAMES = 7;
TB_TXFRAMES = 7;
assign ff_dst = 48'h ABCDEF221100;
assign gm_pquant = 2;
assign gm_pause_gen = rxframe_cnt == 2|rxframe_cnt == 5 ? 1'b1 : 1'b0;
assign gm_vlan_en = rxframe_cnt == 6 ? 1'b1 : 1'b0;
```

Figure 6 shows the reproduction of this test case when the simulation is run. When the MAC function receives the pause frames on its receive path, the MAC function stops transmitting on `gm_tx_d` after TX frame 3 and TX frame 5 for a period of time before resuming the transmission.

Figure 6. Timing Diagram for Test Case 3



Test Case 4

In this test case, the MAC function operates at 1000 Mbps with external loopback. The MAC receive datapath is disabled, and the transmit datapath is enabled. The MAC function transmits two normal Ethernet frames.

By default, the testbench state machine writes to all writable registers during the register configuration. In this test case, the register configuration stops after writing to the `tx_almost_full` register at address offset 0x38.

To reproduce this test case, perform the following steps:

1. Specify the value of parameter `TB_TXFRAMES` to `7;`
2. Look for the control state `stm_typ_wr_tx_af;` and assign the next state to `stm_typ_sim:`

```
stm_typ_wr_tx_af:
begin
if (reg_busy == 1'b 0 && reg_busy_reg == 1'b1)
begin
nextstate <= stm_typ_sim;
end
```

3. In the `if statement`, look for:

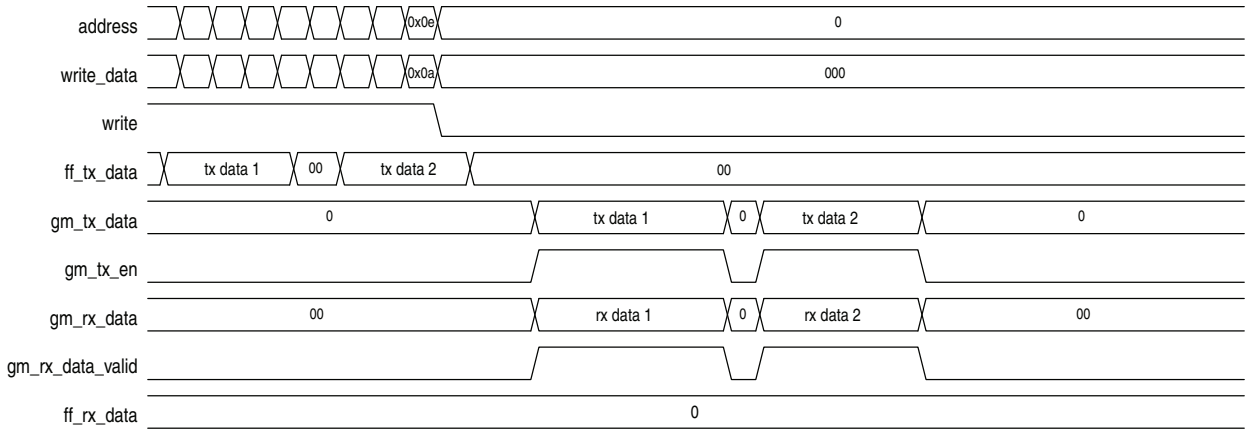
```
else if( nextstate == stm_typ_mac_config)
reg_data_in[1] <=#(2) 1'b 0;
```

and assign `1 b'1` to `reg_data_in[1]` as in the following:

```
else if( nextstate == stm_typ_mac_config)
reg_data_in[1] <=#(2) 1'b 1;
```

Figure 7 shows the behavior of this test case when the simulation is run. The register configuration stops after the `tx_almost_full` register is configured. The MAC function transmits two frames, which are then looped back to the receive path on the GMII. You detect the two frames on `gm_rx_d` but not on `ff_rx_data` because during register configuration, the `rx_en` bit of the `command_config` register is set to `0`.

Figure 7. Timing Diagram for Test Case 4



Test Case 5

In this test case, the MAC function transmits packets with payload length of zero, and receives error packets with no payload from the cable.

The receiver frame format has the following field sizes:

- Preamble = 7 bytes
- Start frame delimiter (SFD) = 1 byte

- Destination address = 6 bytes
- Source address = 6 bytes
- Payload length = 2 bytes
- CRC = 4 bytes

To reproduce this test case, perform the following steps:

1. To transmit packets with payload length of zero, set the following parameter values:

```
TB_LENSTEP = 0;
TB_LENSTART = 0
```

2. To send error packets to the MAC function, configure the Ethernet generator on the GMII as indicated:

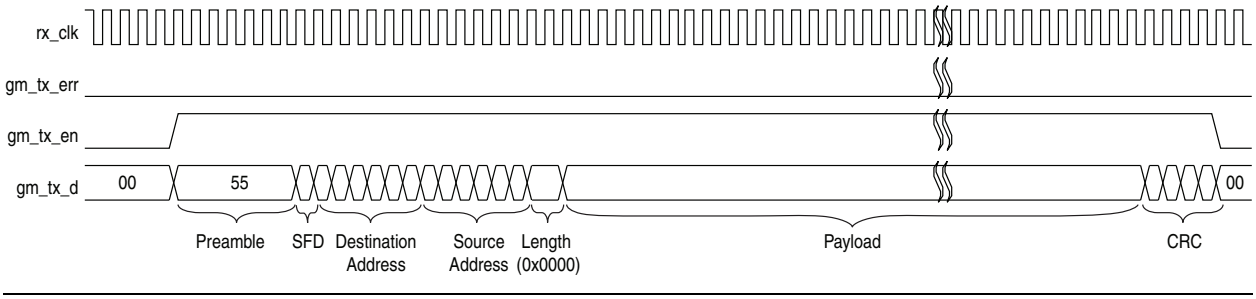
```
assign gm_pad_en = 0;
assign gm_len <= 0
```

Ensure that the `TB_RXFRAMES` parameter's value is more than zero for the Ethernet frame generator on the MII/GMII/RGMII to generate frames. When the `TB_RXFRAMES` parameter equals zero, a loopback is enabled causing the Ethernet frame generator on the MII/GMII/RGMI not to generate frames to the DUT.

Figure 8 and Figure 9 show the reproduction of this test case when the simulation is run.

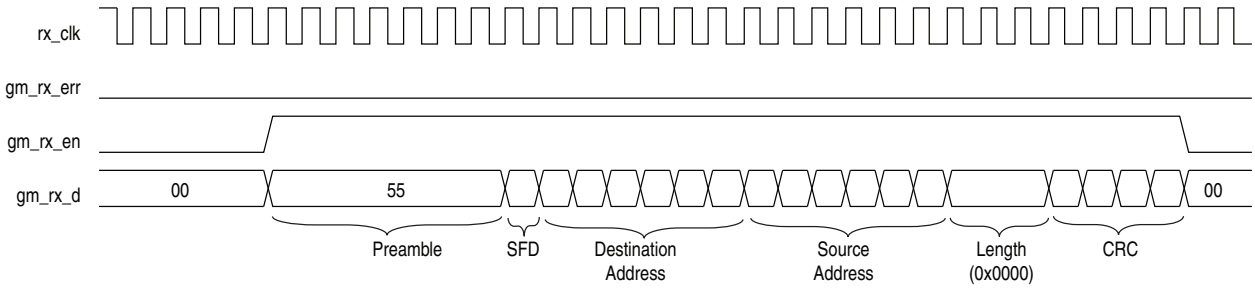
In Figure 8, the packet with zero payload length is detected on `gm_tx_d`. The field length of the packet is 0×0000. The MAC function inserts bytes of zeroes to meet the frame length minimum requirement of 64 bytes.

Figure 8. Timing Diagram for Test Case 5: Transmitter Frame



In Figure 9, an erroneous packet with no payload is received on gm_rx_d.

Figure 9. Timing Diagram for Test Case 5: Receiver Frame



Test Case 6

In this test case, the MAC function receives packets in the following sequence: normal packet with no error; packet with CRC error, followed by packet with a length that exceeds the maximum packet length setting.

To reproduce this test case, perform the following steps:

1. Set the TB_MACLENMAX parameter to the following maximum length configuration for the MAC function:

   ```
   TB_MACLENMAX = 200;
   ```

2. Specify the following packet length value of packet 1:

   ```
   assign gm_len = 160;
   ```

3. Set the subsequent packet lengths to increment by 10 bytes by specifying the following TB_LENSTEP parameter:

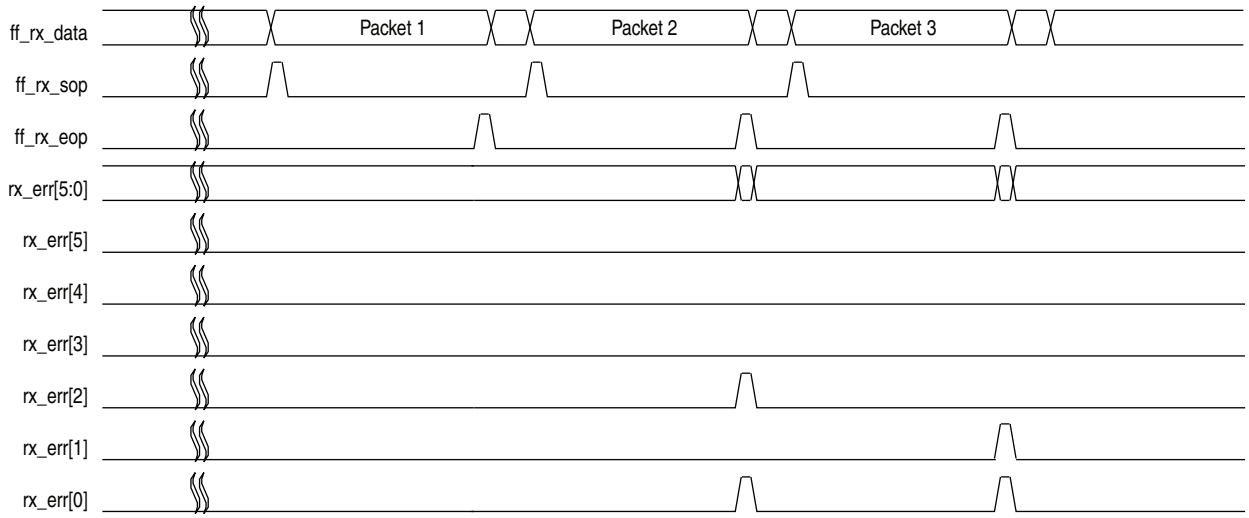
   ```
   TB_LENSTEP = 20;
   ```

4. Set the following zero based counter, rxframe_cnt, to generate CRC error on the second frame:

   ```
   assign gm_crc_error = rxframe_cnt = 1? 1'b1 : 1'b0
   ```

Figure 10 shows the reproduction of this test case when the simulation is run. The `rx_err[2]` signal is asserted at the end of the second receiver packet when the CRC error is detected. The `rx_err[1]` signal is asserted in the third frame to indicate that an invalid frame length is detected.

Figure 10. Timing Diagram for Test Case 6



Conclusion

This application note provides ways to accelerate the debugging process using the Triple Speed Ethernet testbench. By configuring the testbench parameters and states, and reproducing test cases, you can make comparisons between the expected and abnormal signal behaviors.

Document Revision History

Table 4 shows the revision history for this application note.

Table 4. Document Revision History

Date and Revision	Changes Made	Summary of Changes
August 2009, version 1.0	Initial Release.	—

101 Innovation Drive
San Jose, CA 95134
www.altera.com
Technical Support
www.altera.com/support