Firmware Design Document

Module Name: Receive Engine

Author: Zheng Cao
Project Leader: Zheng Cao
Date: 28th November 2005

This page is left intentionally blank.

Revision History

Date	Author	Issue	Comment
28/11/2005	Zheng Cao	1.0	First Version. No removing PAD function

Reference

[1] 10G Ethernet Mac System Design Issue 1.0
[2] Xilinx LogiCORE 10-Gigabit Ethernet MAC User Guide
[3] IEEE 802.3ae Media Access Control (MAC) Parameters, Physical Layers, and Management Parameters for 10 Gb/s Operation

Contents

List of Tables

List of Figures

1 Introduction

The document describes the design of the receive engine used in the Opencores 10-Gigabit Ethernet project. This design is designed to 10-Gigabit Ethernet IEEE 802.3 ae-2002. It is essentially a faster version of the Ethernet where half duplex operation mode is not supported.

The MAC design is loosely based on the Xilinx LogiCORE 10-Gigabit Ethernet MAC, where the transmitter and the receiver incorporate the reconciliation layer. Therefore the receive engine will be specifically designed to interface the client and the physical layer.

2 Detailed Design

2.1 Module Description

The Receive Engine provides the interface between the client and physical layer. Figure 2-1 shows a block diagram of the receive engine with the interfaces to the client, physical, management and the flow control.

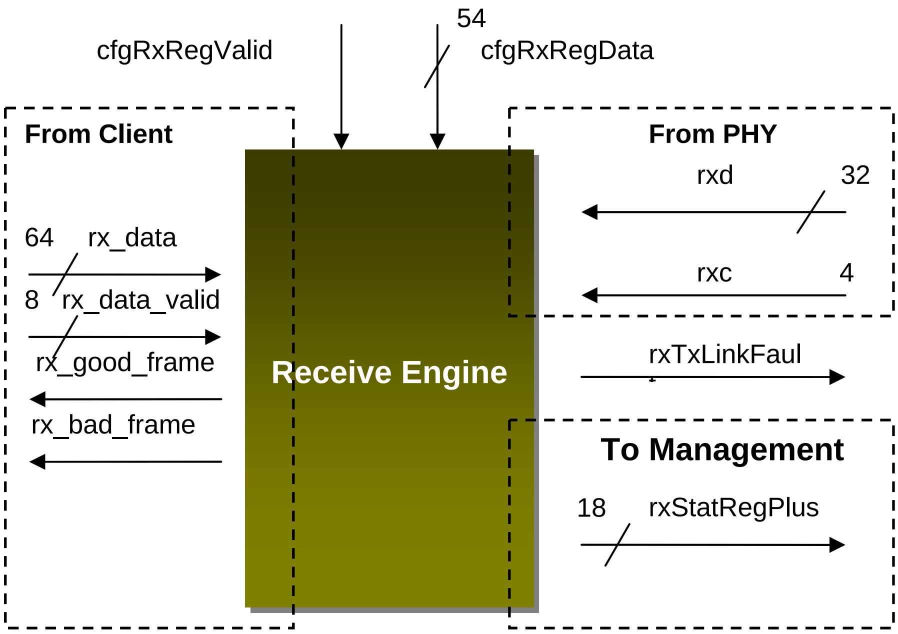

Figure 2-1 Diagram of the Receive Block

2.2 Module Ports

Table 2-1 lists the I/Os that interface to the client.

Port Name	Direction	Description
rx_data[63:0]	Output	Received data, eight bytes wide
rx_data_valid[7:0]	Output	Receive control bits, one bit per lane
rx_good_frame	Output	Asserted at the end of frame to indicate the frame was successfully received and should be processed by user logic
rx_bad_frame	Output	Asserted at the end of frame to indicate the frame was not successfully received and should be discarded by the client

Table 2-1 Client-side interface

Host side Timing sequence of Normal Frame Reception (the same with Xilinx's 10G MAC)

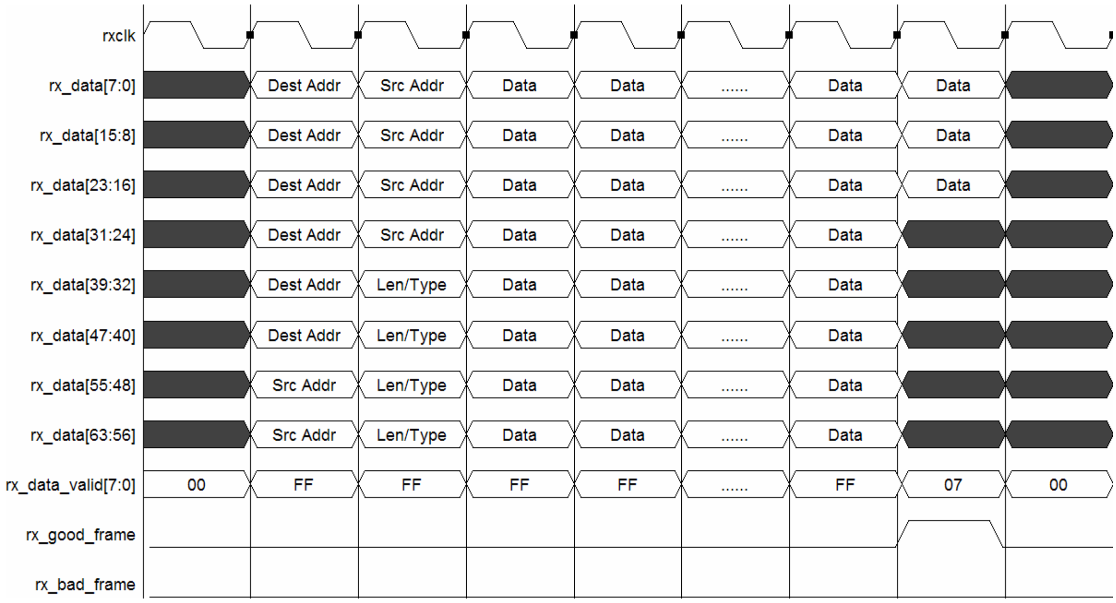

Figure 2-2 Diagram of the Normal Frame Reception

Although the timing diagram in Figure 2-2 shows the rx_good_frame signal asserted at the same time as the last valid data on rx_data, this is not always the case. The rx_good_frame and rx_bad_frame signals are only asserted when all frame checks are completed. If In-band FCS is enabled, then this can be up to five clock cycles after the last valid data is presented; when In-band FCS is disabled, it will be 6 clock cycles. This is represented in Figure 2-3.

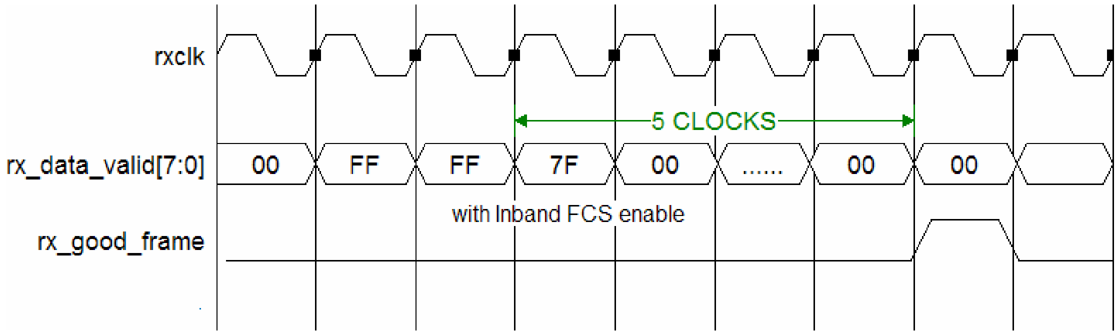

Figure 2-3 rx_good_frame signal timing

Table 2-2 lists the I/Os that interface to PHY.①

Port Name	Direction	Description
xgmii_rxd[31:0]	Input	Received data from PHY. 32bits in 4 lanes.
xgmii_rxc[3:0]	Input	Receive control from PHY. one bit per lane
xgmii_rxclk	Input	Receive clock from PHY. 156.25MHZ

Table 2-2 PHY-side interface

Figure 2-4 is their timing diagram.

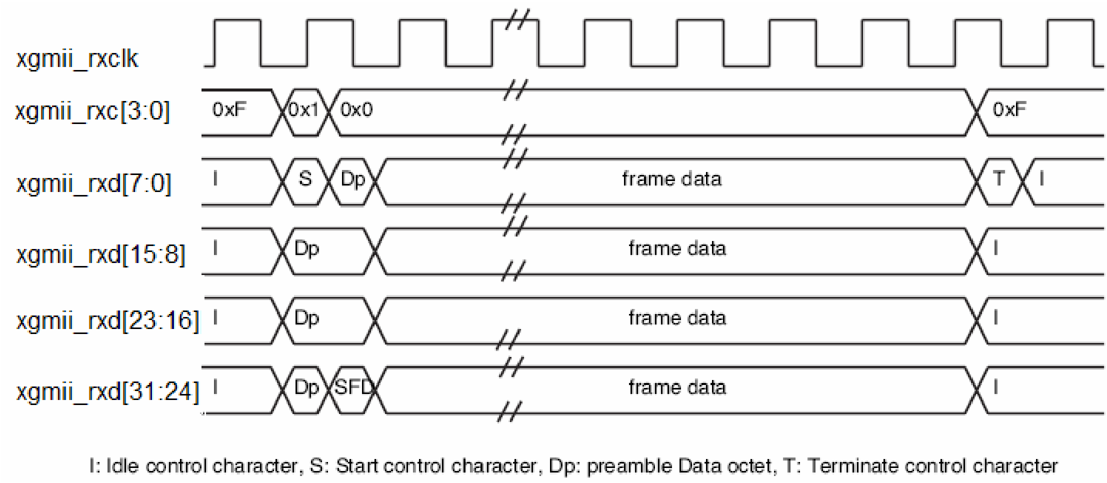

Figure 2-4 Diagram of XGMII receive side signals

Table 2-3 lists the I/Os that interface to management submodule.

Port Name	Direction	Description
cfgRxRegData[52:0]	Input	The value from configure registers. Include both receive and part of reconciliation configure information. See table 2-4
rxStatRegPlus [17:0]	Output	Each bit presents an add operation to a statistic register. These signals should only last for one cycle. For example, when rxStatRegPlus [0] is asserted for one cycle, the counter of Control Frames Received OK register in Management Module will plus one.

Table 2-3 Management-side interface

Table 2-4 lists the signals included in cfgRxRegData.

Signal Name	Direction	Description
cfgRxRegData[47:0]	Input	Multicast address of Receive Engine
cfgRxRegData[48]	Input	When asserted, VLAN mode is enable
cfgRxRegData[49]	Input	When asserted, Receive engine can receive frames.
cfgRxRegData[50]	Input	When asserted, FCS field will be passed to client; when disserted, FCS filed will not be provided to client.
cfgRxRegData[51]	Input	When asserted, Jumbo frame (>9k bytes) will be accepted.
cfgRxRegData[52]	Input	When asserted, Receive Engine will be in reset status.

Table 2-4 Definition of configuration signals

Table 2-5 lists the I/Os that interface to Transmit Engine.

Port Name	Direction	Description
rxTxLinkFault	Output	Indicate that Receive Engine has received Local Link Fault.

Table 2-5 Transmit-side interface

2.3 Module Design

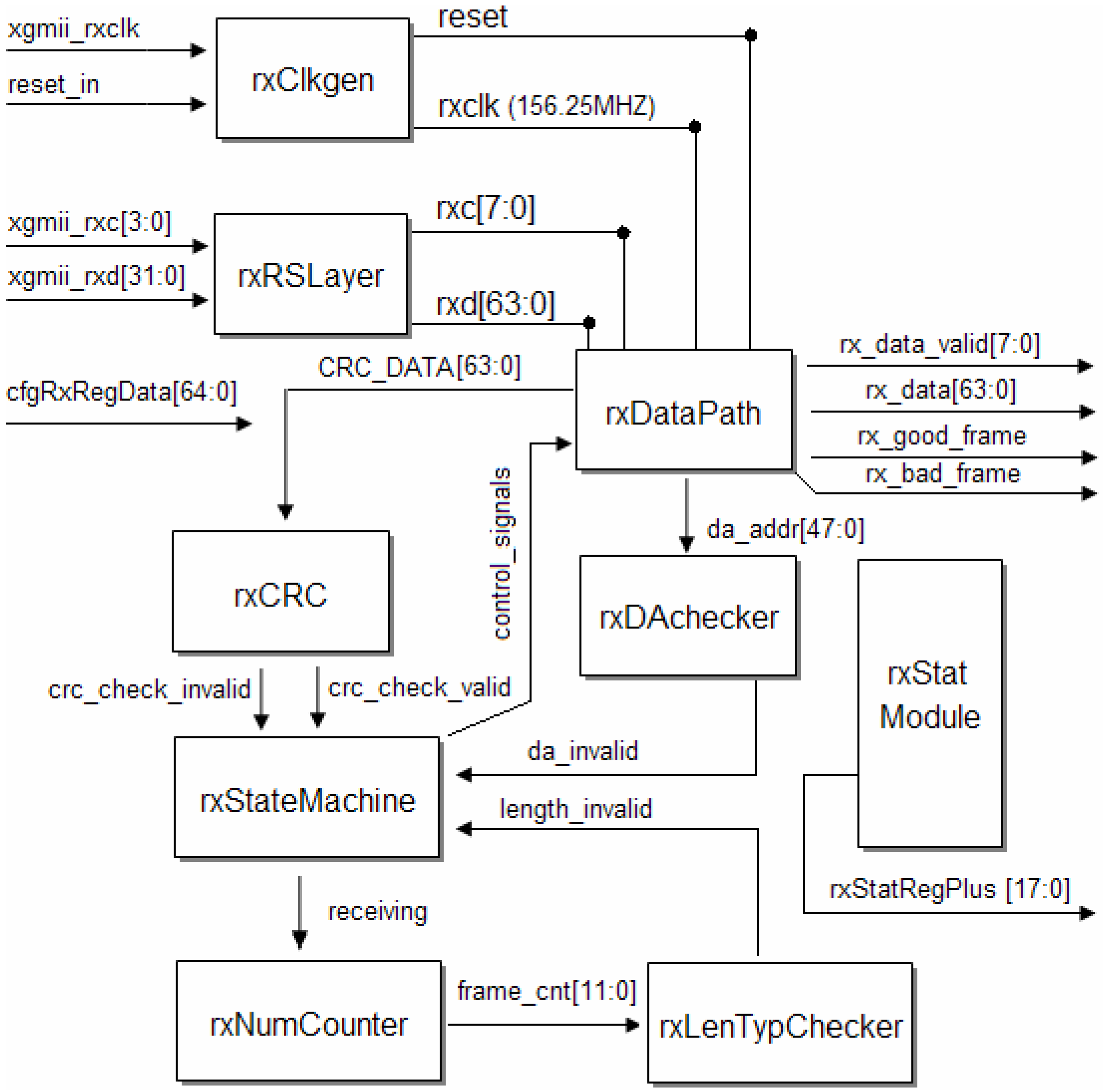

Figure 2-5 internal structure of the Receive Block

Receive Engine is implemented with nine sub-modules, which are:

- **rxClkgen**

  This sub-module is used to generate internal reset and clock signals. In this design, DCM is used. Reset signal is generated from DCM locked signal.

- **rxRSLayer**

  This sub-module implements the receive side function of Reconciliation Sublayer, which is define in IEEE 802.3ae Clause 46. This module samples xgmii_rxd and xgmii_rxc on both rising edge and falling edge of xgmii_rxclk. Besides, this module implements Link Fault Signalling (defined in IEEE 802.3ae Clause 46.3.4).

- **rxDataPath**

  This sub-module is the main data path of Receive Engine. It has functions listed below:

  1. Implements data pipeline;
  2. Indicates SFD, EFD, and Error characters;
  3. Gets Destination Address field and Length/Type field;
  4. Indicates tagged and pause frame by checking Length/Type field;
  5. Generates rx_good_frame and rx_bad_frame signals properly.
  6. Manages Receive FIFOes, FIFOes are:
     a) **Data FIFO:** Data FIFO is used to store valid data from receiving frame. It is a 4K Bytes FIFO, which can store at least two frames.
     b) **Control FIFO:** Control FIFO is used to store control signals, which is 512 Bytes. One bit in Control FIFO presents one byte in Data FIFO. If the bit is '1', then its corresponding Byte is valid. If the bit is '0', then the corresponding Byte is invalid.

- **rxCRC**

  This sub-module implements frame CRC checker. Its generating polynomial is:

  $$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2}+x+1$$

  It is composed of two CRC modules. One is 64bit input width, and the other one is 8bit, both are generated from easics (http://www.easics.be/webtools/crctool). After checking the last byte of frame, if the magic sequence 32'hc704dd7b is generated, then the signal crc_check_valid is asserted; if the magic sequence is not generated, then the signal crc_check_invalid is asserted.

- **rxDAchecker**

  This sub-module is used for check destination address of current frame. It checks four different types of destination address.

  - **Individual address**: the MAC address of this MAC controller.
  - **Broadcast address**: the broadcast address, whose destination address field is filled with all '1's.
  - **Multicast address:** the multicast address, which can be configured by user logic. This reversed MAC address is 01-80-C2-00-00-01. Control Frames will be sent with this destination address.

  - ■ **Other address:** the address of other stations. A frame which carries these destination addresses will be discarded.

  local_invalid signal is asserted when destination address lies in other address.

- ■ **rxStateMachine**

  This sub-module implements state machine of Receive Engine. There are six states:

  - ◆ **IDLE:** Initial status. Controller starts receiving process when SFD received (get_sfd).
  - ◆ **rxReceiveDA:** In this state, controller receives DA field.
  - ◆ **rxReceiveLT:** In this state, controller receives Length/Type field.
  - ◆ **rxReceiveDATA:** In this state, controller receives DATA field. Besides, it also watches DA invalid and Length invalid signals. Any invalid signals will turn state machine to rxGetError state. If no error occurs in receiving states, controller will turn to rxIFGWait.
  - ◆ **rxGetError:** In this state, controller stop receiving, dessert receiving signal (assert when controller is in rxReceiveDA, rxReceiveLT and rxReceiveDATA status) and return controller to IDLE state.
  - ◆ **rxIFGWait:** It is somewhat like a turn around state. It depends on the defined minimum gap between frames.

- ■ **rxNumCounter**

  This sub-module is used for counting frame length. It just is a counter.

- ■ **rxLenTpyChecker**

  This sub-module is used for checking current frame's length. It takes three different situations into account: normal frame, tagged frame and jumbo frame.

  - ■ **Normal Frame:** Minimum Length: 64bytes; Maximum Length: 1518
  - ■ **Tagged Frame:** Minimum Length: 64bytes; Maximum Length: 1522
  - ■ **Jumbo Frame:** Minimum Length: 64byte; Maximum Length: 9K

- ■ **rxStatModule**

  This sub-module is used to collect statistic information of MAC controller. See Management Module datasheet for detailed information of statistics.

2.4 Block Diagram

In this section, diagram of each sub-modules will be listed with some descriptions.

rxRSLayer

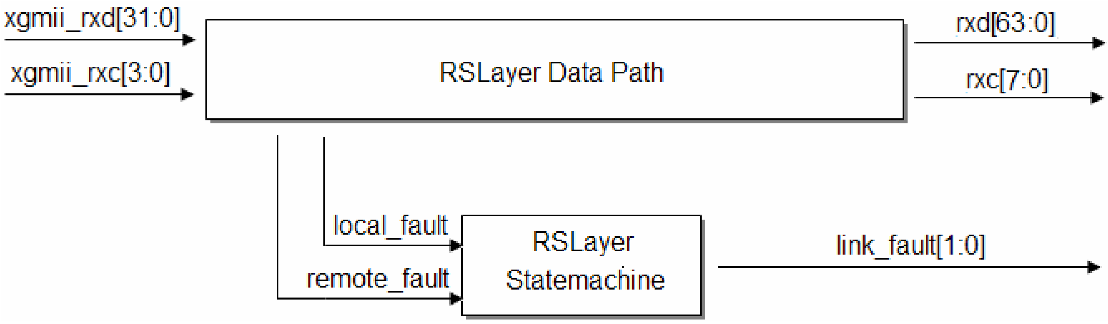

Figure 2-6 block diagram of rxRSLayer

rxDAchecker

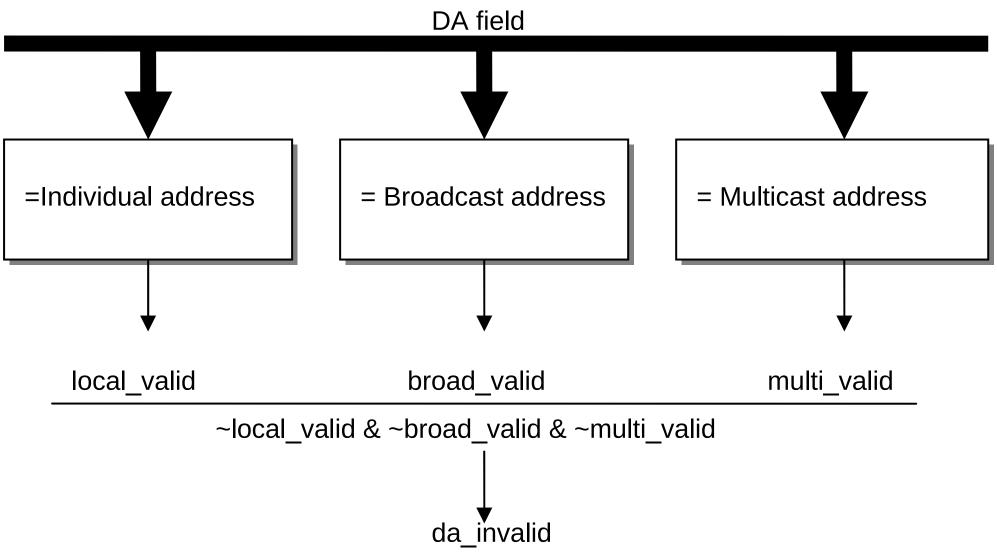

Figure 2-7 block diagram of rxDAChecker

rxLenTypChecker

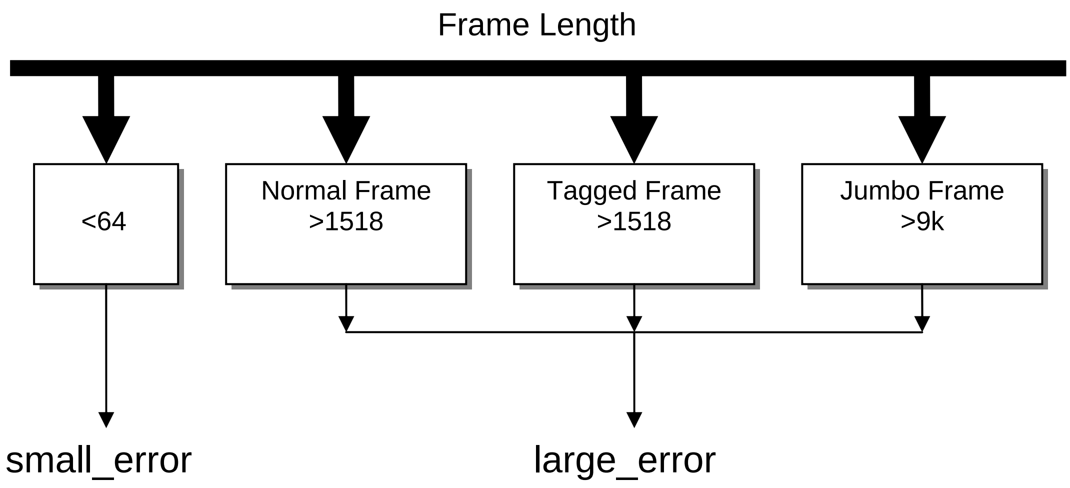

Figure 2-8 block diagram of rxLenTypChecker

rxCRCChecker

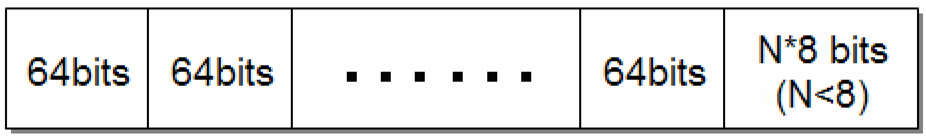

Figure 2-9 10Gigabit Ethernet Frame division

For data which is full of 64bits, it uses 64bit CRC Module to generate CRC value. For last data which is not full of 64bits, it uses 8bit CRC Module to generate CRC value.

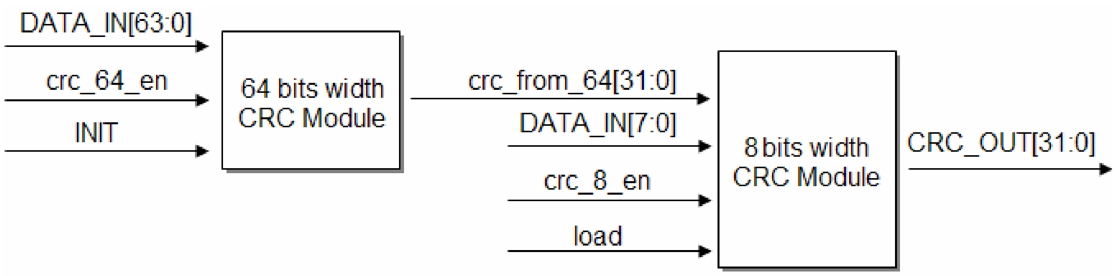

Figure 2-10 block diagram of rxCRCChecker

rxDataPath

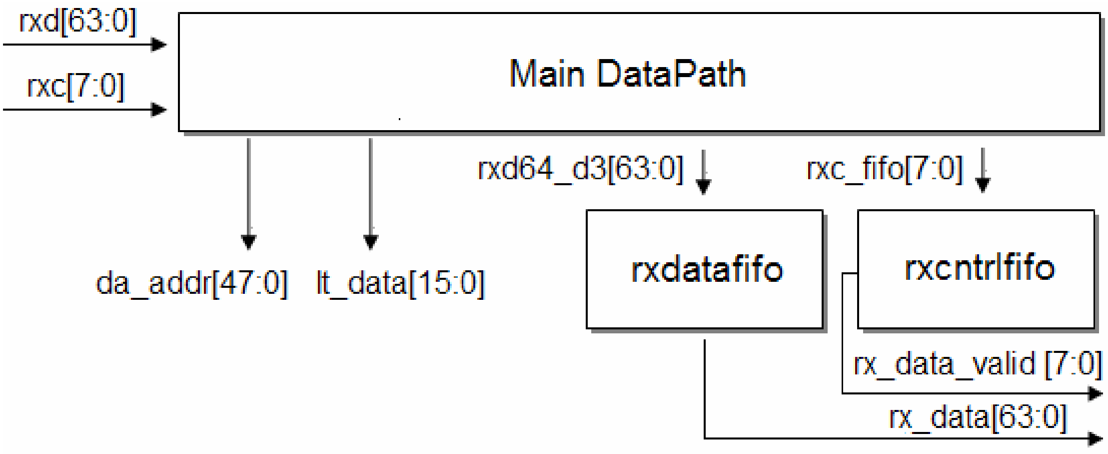

Figure 2-11 block diagram of rxDataPath

rxStateMachine

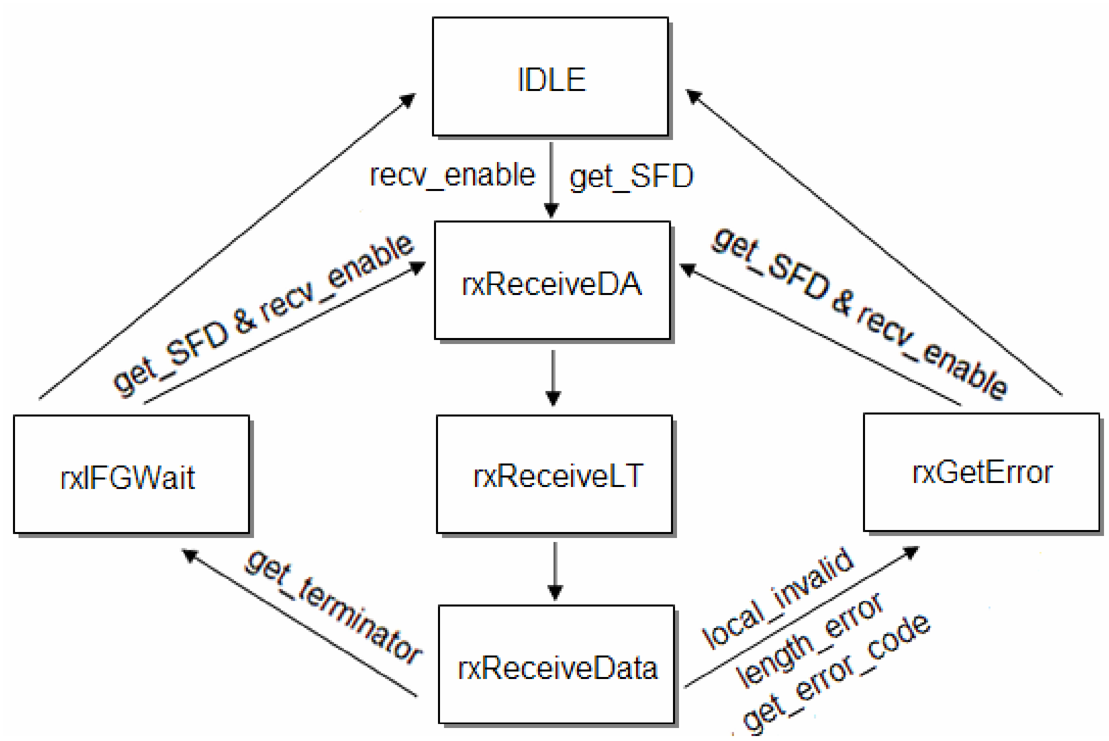

Figure 2-12 block diagram of rxStateMachine

2.5 Code Listing

Source Code Name	Version	Date
timescale.v	1.0	
xgiga_define.v	1.0	
rxReceiveEngine.v	1.0	
rxRSLayer.v	1.0	
rxRSIO.v	1.0	
rxLinkFaultState.v	1.0	
rxDataPath.v	1.0	
rxClkgen.v	1.0	

rxNumCounter.v	1.0	
rxDAchecker.v	1.0	
rxLenTypChecker.v	1.0	
rxCRC.v	1.0	
rxStateMachine.v	1.0	
rxStatModule.v	1.0	
CRC32_D8.v	1.0	
CRC32_D64.v	1.0	
rxdatafifo.v	1.0	
rxcntrlfifo.v	1.0	

Table 2-6 Code Listing

3 Traceability Matrix

802.3ae Clause	Implemented In
1	

Table 3-1 Traceability Matrix

4 Abbreviation

FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
PHY	Physical
UML	Unified Modelling Language