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 considerations of statistics support.

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.

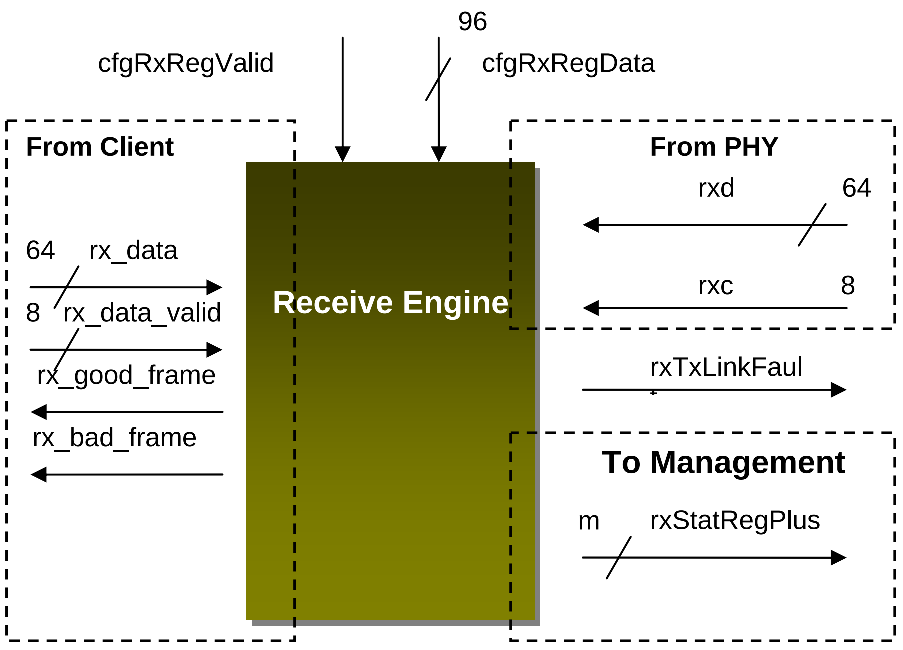


Figure 2-1 Diagram of the Receive Block

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 user logic

Table 2-1 Client-side interface

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

Port Name	Direction	Description
rxd[63:0]	Input	Received data from PHY. The format of data frame is the same as transmit engine transmit.
rxc[7:0]	Input	Receive control from PHY. one bit per lane

Table 2-2 PHY-side interface

Note 1: Details will be showed in Appendix (Time sequence of signals).

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

Port Name	Direction	Description
cfgRxRegData[67:0]	Input	The value from configure registers. Include both receive and part of reconciliation configure information.
cfgRxRegValid	Input	When it is asserted, it indicates cfgRxRegData is valid.
rxStatRegPlus [n-1:0]	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 I/Os that interface to Transmit Engine.

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

Table 2-4 Transmit-side interface

2.3 Module Design

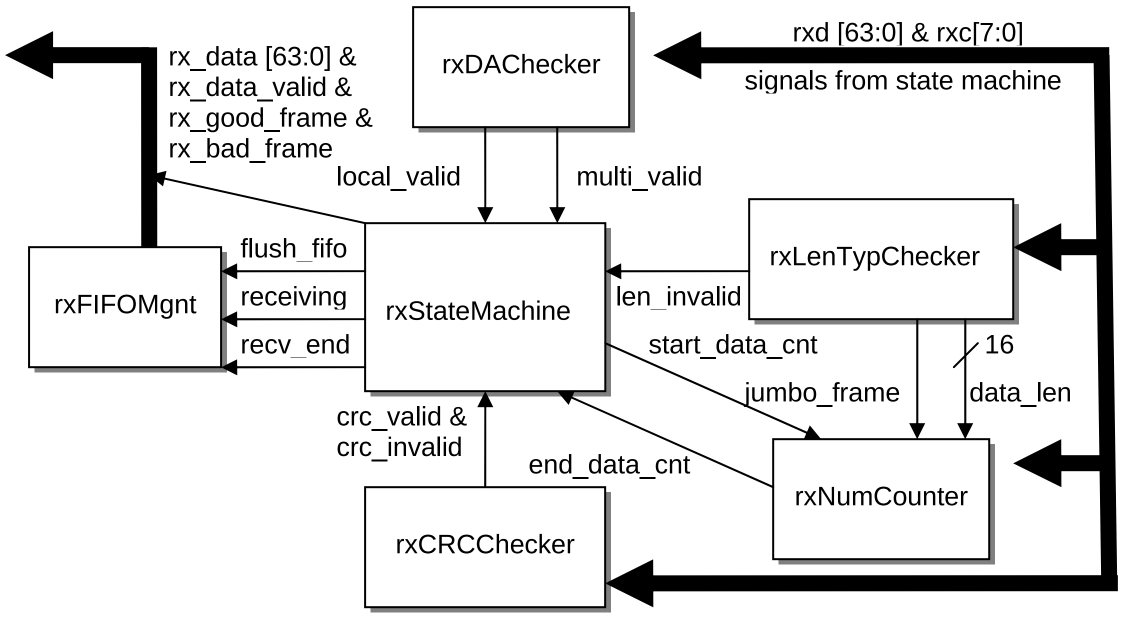


Figure 2-2 internal structure of the Receive Block

Receive Engine is implemented with six sub-modules. The structure of Receive Engine is showed in Figure 2-2 above.

Functions of sub-modules will be listed below:

- **rxDAchecker**

  This sub-module is used for check destination address of receiving frame. There are four different types of destination address.

  1. **Individual address**: the MAC address of this station
  2. **Broadcast address**: the broadcast address, which destination address field is filled with all ‘1’s.
  3. **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.
  4. **Other address:** the address of other stations. A frame which carries these destination addresses will be discarded.

  A valid destination address must be in 1, 2 or 3 category.

- **rxLenTpyChecker**

  This sub-module is used for check Length/Type field of receiving frame. It can be divided into four groups:

  1. **Padded length:** The value of Length/Type field is less than 64. It indicates that current frame is a padded frame. This actual length will be used in *remove PAD process*.
  2. **Normal length:** The value of Length/Type field is more than 64, less than 1518. It indicates that current frame is with valid length. Its length will be used in *rxNumCounter* module to control *data field receiving process.*
  3. **Jumbo length:** The value of Length/Type field is more than 1518. If jumbo frame is enabled by user logic, then MAC will receive this frame, or it will be treated as an invalid frame, and report length error to *rxStateMachine* module.
  4. **Type:** If value of Length/Type field is more than 1536, then this field may be a type interpretation. This core supports two types: 0x8100 (tagged frame) and 0x8808 (pause frame).

  Without jumbo frame enable configuration, frames whose Length/Type value are more than 1518(1522 if tagged frame) will be treated as length error. Such report will send to rxStateMachine.

- **rxCRCChecker**

  This sub-module is used for checking FCS field of a receiving frame. It is a 32bit CRC module, whose implementation takes xilinx xapp209 as reference. The generating polynomial is

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

- **rxNumCounter**

  This sub-module is used for counting true frame length and indicating the end of *data field receiving process*. In this sub-module, it counts three numbers:

  1. **Data field length:** It can be 46, (Length/Type – 18) or tagged frame length.
  2. **True data field length:** It can be (Length/Type - 18) or tagged frame length.
  3. **Total frame length:** It is the total length from DA field to FCS field.

  If total frame length is not equal to Length/Type or 64 bytes, then a length error report will be generated. The receiving frame will be indicated as a bad frame.

  This module should also deal with another challenge, which is about reception of error propagation.

  During receiving period, whenever we receive Error Character, the counter should be delayed or paused.

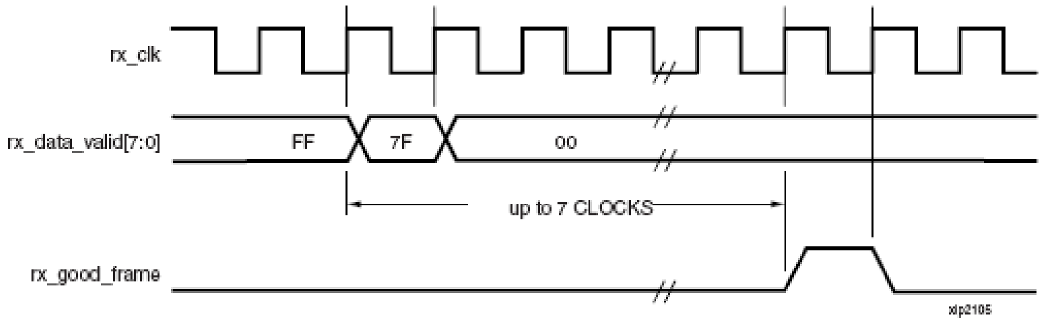
- **rxFIFOMgnt**

  This sub-module is used for managing receive FIFO and interface to user logic. There are two kinds of FIFO in this sub-module:

  1. **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.
  2. **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.

  The two FIFO are synchronizing with receive clock. They begin to store data when *receiving* signal is set by *Receive State Machine,* and stop storing when *recv_end* signal is set by *Receive State Machine*.

FIFOs are always providing data to user logic unless they are empty. If there are errors, such as DA invalid, Length invalid and FCS invalid, the frame will be indicated as bad frame up to 7 clocks after last data transfer. So does good frame.



- **rxStateMachine**

  This sub-module is the main part of Receive Engine. There are six states:

  1. **IDLE:** Initial status. In this state, it only respond to get_sfd signal (indicate that a SFD has been received).
  2. **rxReceiveDA:** In this state, MAC begins to check DA field.
  3. **rxReceiveLT:** In this state, MAC begins to check Length/Type field.
  4. **rxReceiveDATA:** In this state, MAC begins to receiving DATA field. Besides, it also watches DA invalid and Length invalid signals. Any invalid signals will lead the state machine to rxWaitCheck state, and drive recv_end signal to '1' (stop receiving current frame).
  5. **rxReceiveFCS:** In this state, MAC begins to receiving FCS field.
  6. **rxWaitCheck:** In this state, it drives recv_end signal to '1'. Any error signals will be translated to rx_bad_frame signal, and good signals will be translated to rx_good_frame signal.

Detailed description will be listed in section 2.4.

2.4 Block Diagram

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

rxDAchecker

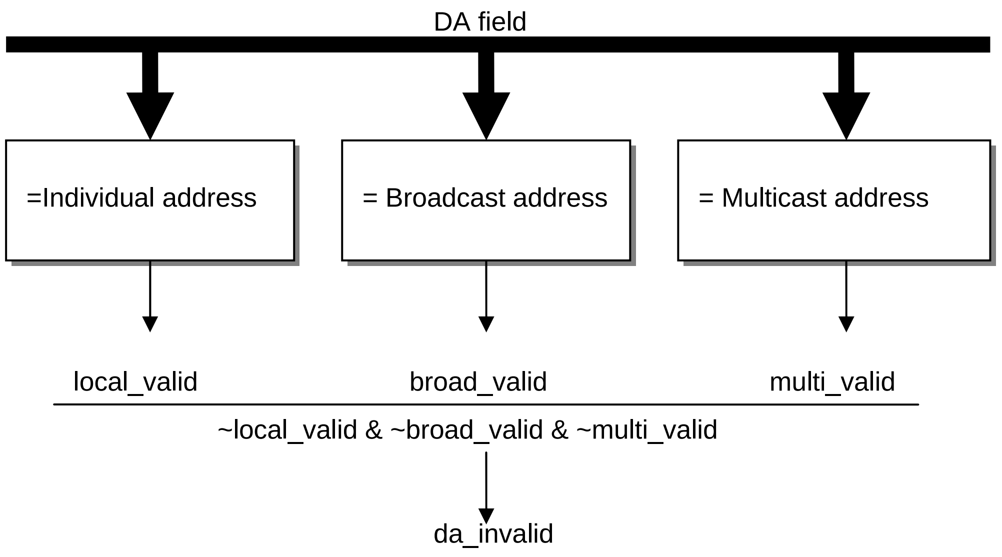


Figure 2-3 block diagram of rxDAChecker

rxLenTypChecker

`

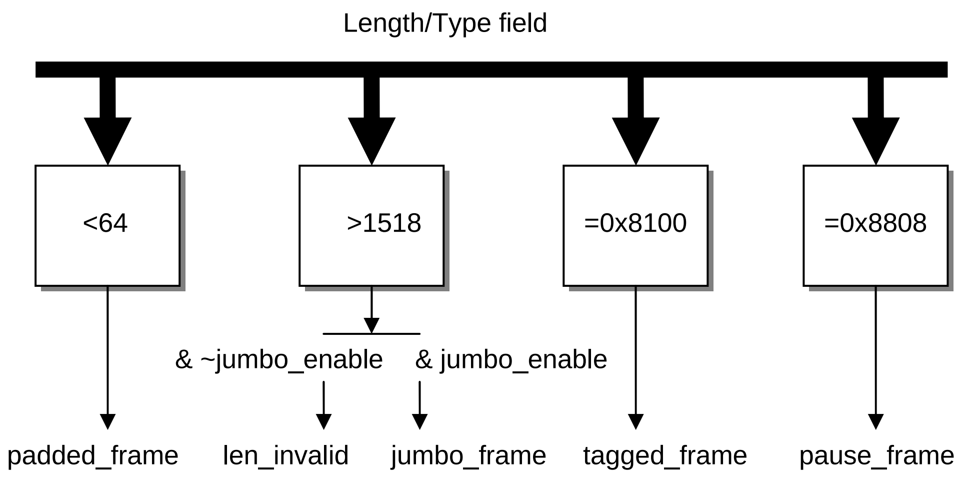


Figure 2-4 block diagram of rxLenTypChecker

rxCRCChecker

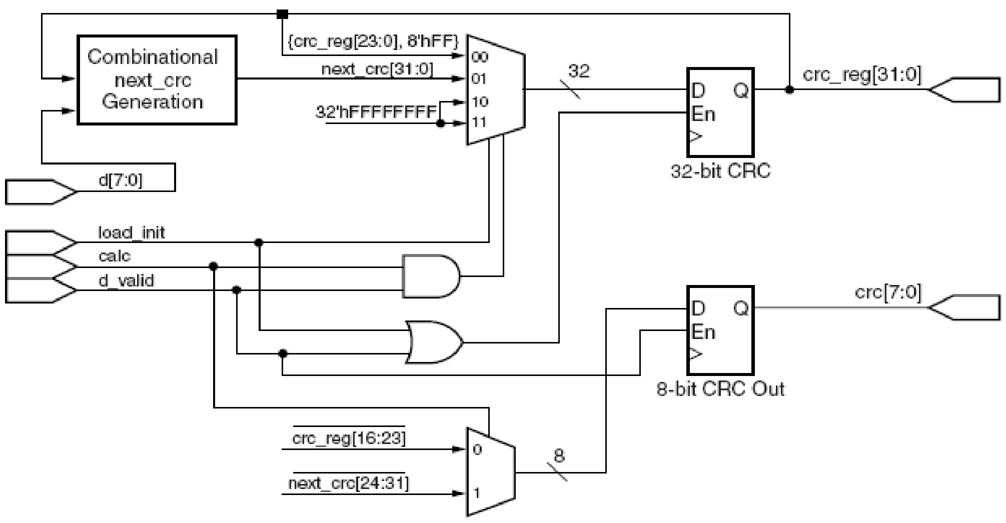


Figure 2-5 block diagram of rxCRCChecker

rxNumCounter

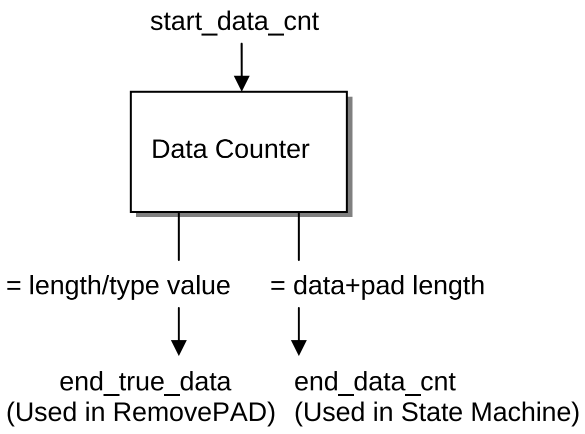


Figure 2-6 block diagram of rxNumCounter

Actually, this sub-module is much more difficult than the diagram shows. Because input data is 64bit width, but the counter is counting in byte. A good and detailed diagram will be pasted later (not very easy to draw out).

rxFIFOMgnt

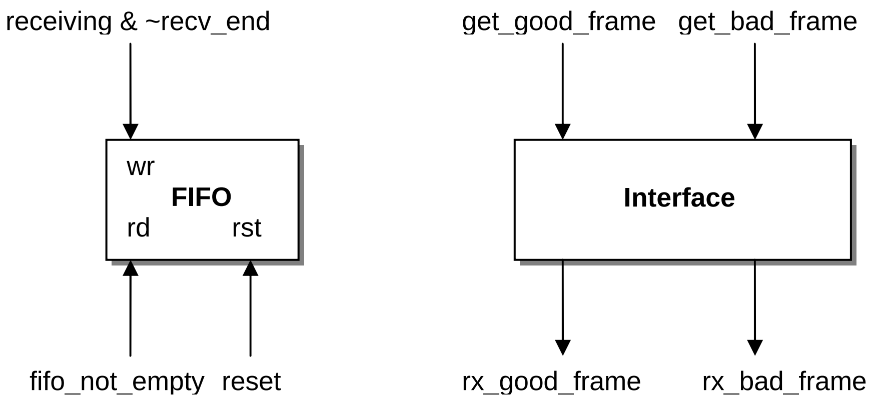


Figure 2-7 block diagram of rxFIFOMgnt

rxStateMachine

This is the main part of Receive Engine. All control signals are send out from it. Figure 2-8 is a brief state machine of Receive Engine, which only takes normal data frame into account. The real state machine implemented now is including tagged frame too.

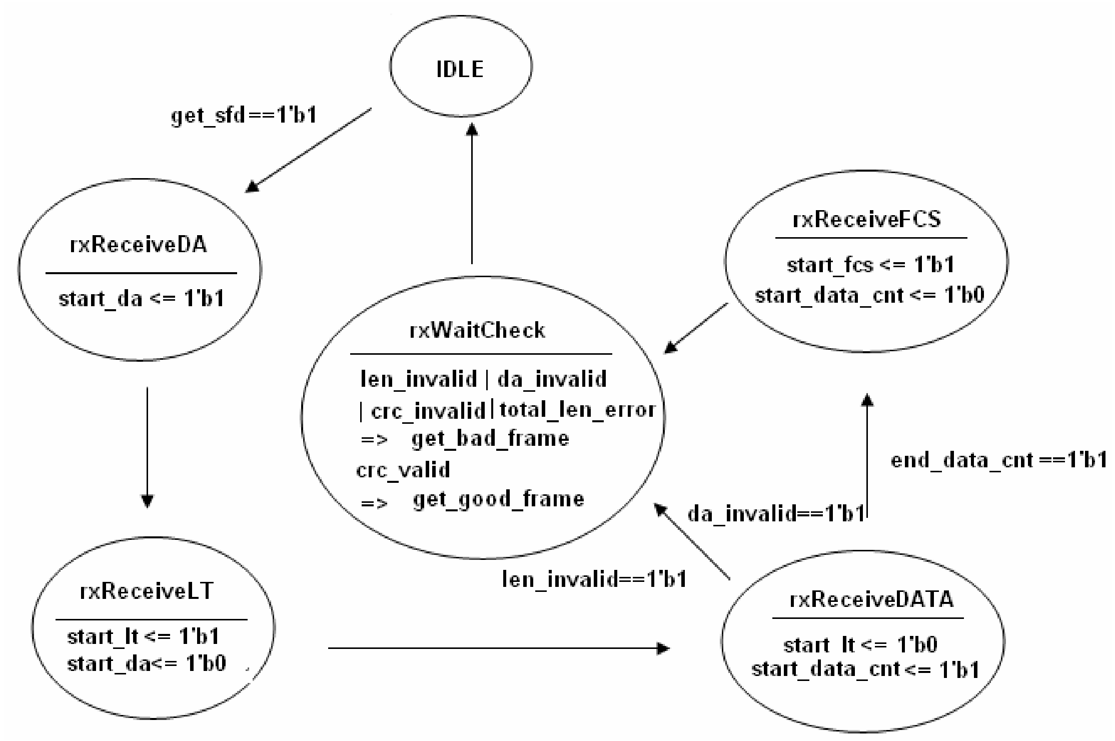


Figure 2-8 block diagram of rxStateMachine

In this sub-module, it also contains another state machine which is used in receiving Control Frame. That state machine will be appended to this document later.

2.5 Code Listing

Source Code Name	Version	Date
	1.0	

Table 2-5 Code Listing

3 Traceability Matrix

802.3ae Clause	**Implemented In**
1	tx_buffer.v

Table 3-1 Traceability Matrix

4 Abbreviation

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