

# IOb-Eth

Ethernet MAC IP core

February 9, 2026

User Guide, V0.1, Build 266854a







## Document Version History

Version	Date	Person	Changes from previous version
0.1	February 9, 2026	AN	Initial document version



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Features . . . . .	2
1.2	Deliverables . . . . .	2
<b>2</b>	<b>Description</b>	<b>2</b>
2.1	Block Diagram . . . . .	2
2.2	Configuration . . . . .	4
2.3	Interface Signals . . . . .	4
2.4	Control and Status Registers . . . . .	8
<b>3</b>	<b>Usage</b>	<b>9</b>
3.1	Instantiation . . . . .	9
3.2	Simulation . . . . .	11
<b>4</b>	<b>Baremetal Drivers</b>	<b>12</b>
4.1	eth_frame_struct.h File Reference . . . . .	12
4.2	eth_mem_map.h File Reference . . . . .	13
4.3	iob_eth.h File Reference . . . . .	13
4.3.1	Detailed Description . . . . .	14
4.3.2	Function Documentation . . . . .	15
4.4	iob_eth_csrs.h File Reference . . . . .	21
4.4.1	Detailed Description . . . . .	23
4.4.2	Function Documentation . . . . .	24
4.5	iob_eth_defines.h File Reference . . . . .	38
4.5.1	Detailed Description . . . . .	38
4.5.2	Macro Definition Documentation . . . . .	39
4.6	iob_eth_macros.h File Reference . . . . .	43
4.6.1	Detailed Description . . . . .	46

4.6.2	Macro Definition Documentation . . . . .	46
4.7	iob_eth_rmac.h File Reference . . . . .	60
<b>5</b>	<b>Linux Software</b>	<b>61</b>
5.1	Kernel Module . . . . .	61
5.2	User Space Interfaces . . . . .	61
5.2.1	/dev Interface . . . . .	62
5.2.2	ioctl Interface . . . . .	63
5.2.3	sysfs Interface . . . . .	65
5.3	User Space Application . . . . .	66
5.4	Tests . . . . .	66
<b>6</b>	<b>CSR Detailed Description for Software</b>	<b>67</b>
<b>7</b>	<b>Buffer Descriptors</b>	<b>73</b>

## List of Tables

1	Core subblocks. . . . .	3
2	General operation group . . . . .	4
3	Clock, clock enable and reset . . . . .	4
4	AXI manager interface for external memory . . . . .	5
5	Interrupt Output . . . . .	5
6	PHY reset output (active low) . . . . .	5
7	MII interface . . . . .	6
8	Control and status interface, when selecting the AXI-Lite CSR interface. . . . .	7
9	Control and status interface, when selecting the Wishbone CSR interface. . . . .	7
10	Control and Status Registers interface (auto-generated) . . . . .	7
11	IOb_Eth Software Accessible Registers . . . . .	8
12	Data transfer/status registers for use without DMA . . . . .	9
13	PHY reset control registers . . . . .	9
14	IOb_Eth Buffer Descriptors . . . . .	9
15	General Registers. . . . .	9
72	MODER (Mode Register) . . . . .	68
73	INT_SOURCE (Interrupt Source Register) . . . . .	69
74	INT_MASK (Interrupt Mask Register) . . . . .	69
75	IPGT (Back to Back Inter Packet Gap Register) . . . . .	69
76	IPGR1 (Non Back to Back Inter Packet Gap Register 1) . . . . .	69
77	IPGR2 (Non Back to Back Inter Packet Gap Register 2) . . . . .	70
78	PACKETLEN (Packet Length Register) . . . . .	70
79	COLLCONF (Collision and Retry Configuration Register) . . . . .	70
80	TX_BD_NUM (Transmit BD Number Register) . . . . .	70
81	CTRLMODER (Control Module Mode Register) . . . . .	71
82	PASSALL and RXFLOW Operation . . . . .	71

83	MIIMODER (MII Mode Register) . . . . .	71
84	MIICOMMAND (MII Command Register) . . . . .	71
85	MIIADDRESS (MII Address Register) . . . . .	72
86	MIITX_DATA (MII Transmit Data) . . . . .	72
87	MIIRX_DATA (MII Receive Data) . . . . .	72
88	MIISTATUS (MII Status Register) . . . . .	72
89	MAC_ADDR0 (MAC Address Register 0) . . . . .	72
90	MAC_ADDR1 (MAC Address Register 1) . . . . .	73
91	HASH0 (HASH Register 0) . . . . .	73
92	HASH1 (HASH Register 1) . . . . .	73
93	TXCTRL (Tx Control Register) . . . . .	73
94	Tx Buffer Descriptor . . . . .	74
95	Rx Buffer Descriptor . . . . .	75



## List of Figures

1	IP Core Symbol . . . . .	1
2	High-Level Block Diagram . . . . .	3
3	Core Instance and Required Surrounding Blocks . . . . .	10
4	Testbench Block Diagram . . . . .	11



# 1 Introduction

IOb-Eth is an Ethernet MAC IP core written in Verilog, which users can download for free, modify, simulate and implement in FPGA. The core supports 10/100 Mbps operation, featuring PHY, MDIO MDC, control and data interfaces. The control interface supports AXI-Lite, Wishbone, and the native IOb bus. The data interface supports AXI4 optionally the data can be driven through the control interface. It can run in baremetal or Linux using the open source Ethmac Linux drivers. The IP core can be synthesized fro both ASICs and FPGAs.

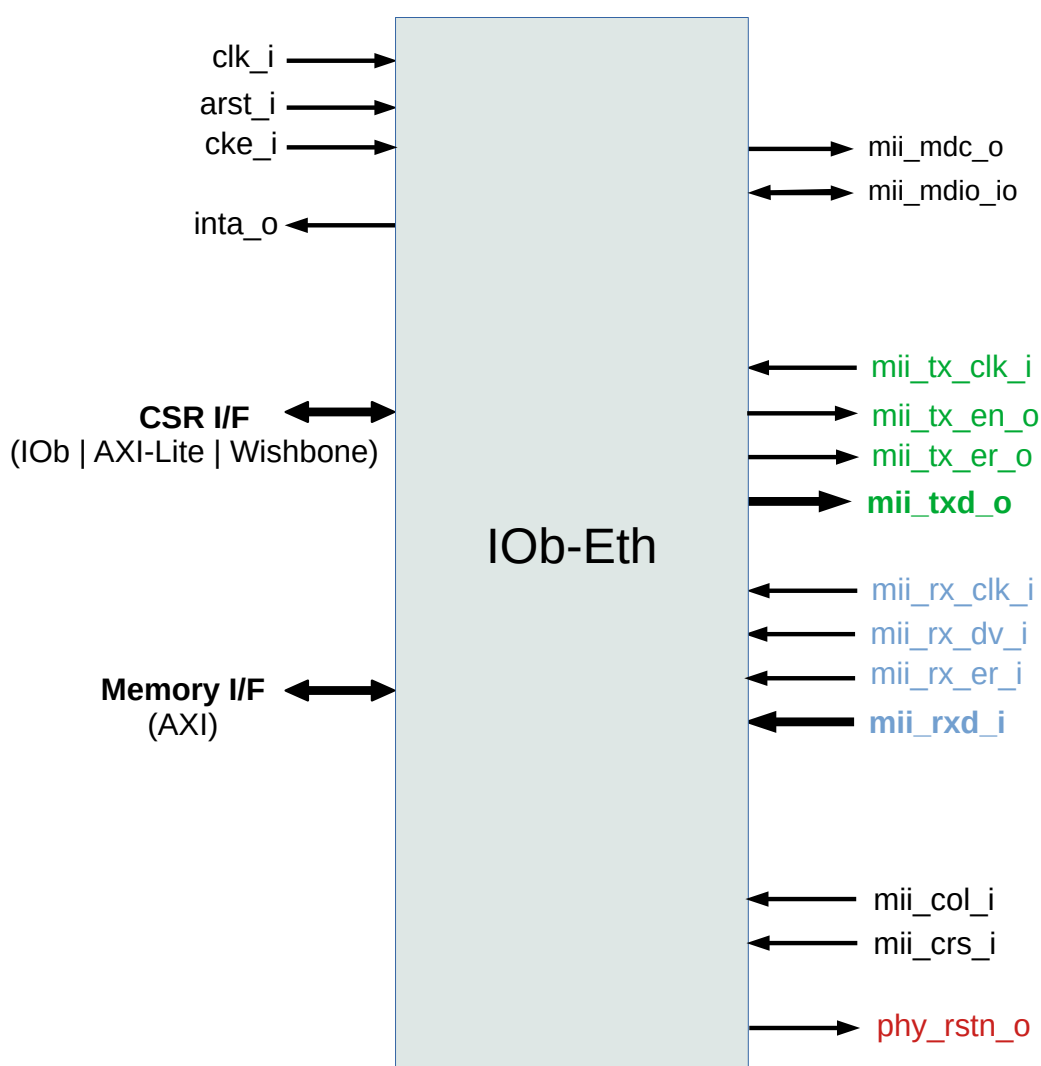


Figure 1: IP Core Symbol

## 1.1 Features

- 10/100 Mbps Operation
- MII interface
- AXI data interface
- Up to 128 Tx/Rx buffer descriptors
- Support for multiple control interface types: AXI-Lite, Wishbone, and IOb
- Features baremetal and Linux software drivers.

## 1.2 Deliverables

- Verilog RTL source code synthesizable for ASIC and FPGA
- Verilog testbench and simulation scripts for code coverage
- ASIC synthesis script and timing constraints
- FPGA synthesis scripts and timing constraints
- Bare-metal software driver and example user firmware
- Comprehensive user guide

## 2 Description

This section gives a detailed description of the IP core. The high-level block diagram is presented, along with a description of its subblocks. The parameters and macros that define the core configuration are listed and explained. The interface signals are enumerated and described; if timing diagrams are needed, they are shown after the interface signals. Finally the Control and Status Registers (CSR) are outlined and explained.

### 2.1 Block Diagram

Figure 2 presents a high-level block diagram of the core, followed by a brief description of each block.

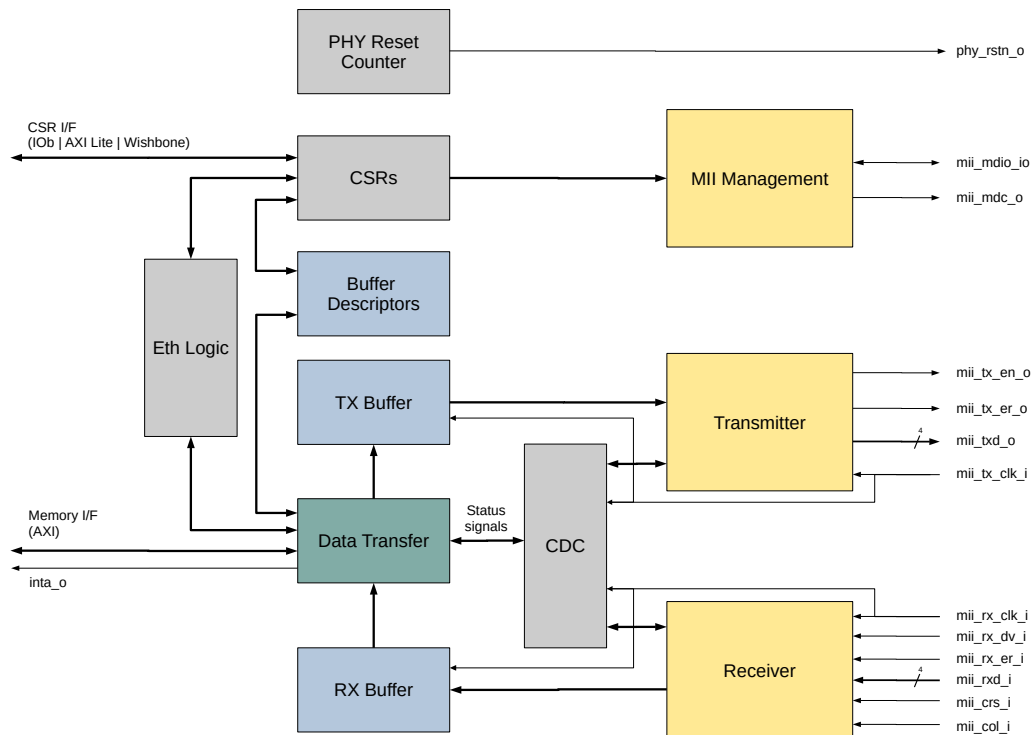


Figure 2: High-Level Block Diagram

The Verilog modules in the top-level entity of the core are described in the following tables. The table elements represent the subblocks in the Block Diagram.

Name	Description
csrs	The Control and Status Register block contains registers accessible by the software for controlling the IP core attached as a peripheral.
phy_reset_counter	Counter to generate initial PHY reset signal. Configurable duration based on counter reset value.
cdc	Clock domain crossing block, using internal synchronizers.
transmitter	Ethernet transmitter that reads payload bytes from a host interface, emits preamble/SFD and payload, computes and appends the CRC, and provides flow-control so the surrounding logic knows when the transmitter is ready for the next frame.
receiver	Ethernet receiver that detects frame start, captures the destination MAC and payload, writes received bytes to a host interface, and validates the frame with a CRC check; it produces a ready/received indication for higher-level logic.
tx_buffer	Buffer memory for data to be transmitted.
rx_buffer	Buffer memory for data received.
buffer_descriptors	Buffer descriptors memory.
data_transfer	Manages data transfers between ethernet modules and interfaces.
mii_management	Controls MII management signals.
eth_logic	Extra ethernet logic for interface between CSRs and Data Transfer block.

Table 1: Core subblocks.

## 2.2 Configuration

The following tables describe the IP core configuration. The core may be configured using macros or parameters:

**'M'** Macro: a Verilog macro or `define` directive is used to include or exclude code segments, to create core configurations that are valid for all instances of the core.

**'P'** Parameter: a Verilog parameter is passed to each instance of the core and defines the configuration of that particular instance.

Configuration	Type	Min	Typical	Max	Description
PREAMBLE	M	NA	8'h55	NA	Ethernet packet preamble value
PREAMBLE_LEN	M	NA	7	NA	Ethernet packet preamble length
SFD	M	NA	8'hD5	NA	Start Frame Delimiter
MAC_ADDR_LEN	M	NA	6	NA	Ethernet MAC address length
DATA_W	P	NA	32	128	Data bus width
AXI_ID_W	P	0	1	32	AXI ID bus width
AXI_ADDR_W	P	1	24	32	AXI address bus width
AXI_DATA_W	P	1	32	32	AXI data bus width
AXI_LEN_W	P	1	4	4	AXI burst length width
PHY_RST_CNT	P	NA	20'hFFFFFF	NA	PHY reset counter value. Sets the duration of the PHY reset signal. Suggest smaller value during simulation, like 20'h00100.
BD_NUM_LOG2	P	NA	7	7	Log2 amount of buffer descriptors
BUFFER_W	P	0	11	32	Buffer size
VERSION	M	NA	16'h0001	NA	Product version. This 16-bit macro uses nibbles to represent decimal numbers using their binary values. The two most significant nibbles represent the integral part of the version, and the two least significant nibbles represent the decimal part.

Table 2: General operation group

There may be other macros or generic parameters in the code that are not documented. They are typically derived from other primary macros or parameters or exist for documentation purposes.

## 2.3 Interface Signals

The interface signals of the core are described in the following tables. Note that the output signals are registered in the core, while the input signals are not.

Name	Direction	Width	Description
clk_i	input	1	Clock
cke_i	input	1	Clock enable
arst_i	input	1	Asynchronous active-high reset

Table 3: Clock, clock enable and reset

Name	Direction	Width	Description
axi_araddr_o	output	AXI_ADDR_W	AXI address read channel byte address.
axi_arvalid_o	output	1	AXI address read channel valid.
axi_arready_i	input	1	AXI address read channel ready.
axi_rdata_i	input	AXI_DATA_W	AXI read channel data.
axi_rresp_i	input	2	AXI read channel response.
axi_rvalid_i	input	1	AXI read channel valid.
axi_rready_o	output	1	AXI read channel ready.
axi_arid_o	output	AXI_ID_W	AXI address read channel ID.
axi_arlen_o	output	AXI_LEN_W	AXI address read channel burst length.
axi_arsize_o	output	3	AXI address read channel burst size.
axi_arburst_o	output	2	AXI address read channel burst type.
axi_arlock_o	output	2	AXI address read channel lock type.
axi_arcache_o	output	4	AXI address read channel memory type.
axi_arqos_o	output	4	AXI address read channel quality of service.
axi_rid_i	input	AXI_ID_W	AXI Read channel ID.
axi_rlast_i	input	1	AXI Read channel last word.
axi_awaddr_o	output	AXI_ADDR_W	AXI address write channel byte address.
axi_awvalid_o	output	1	AXI address write channel valid.
axi_awready_i	input	1	AXI address write channel ready.
axi_wdata_o	output	AXI_DATA_W	AXI write channel data.
axi_wstrb_o	output	AXI_DATA_W/8	AXI write channel write strobe.
axi_wvalid_o	output	1	AXI write channel valid.
axi_wready_i	input	1	AXI write channel ready.
axi_bresp_i	input	2	AXI write response channel response.
axi_bvalid_i	input	1	AXI write response channel valid.
axi_bready_o	output	1	AXI write response channel ready.
axi_awid_o	output	AXI_ID_W	AXI address write channel ID.
axi_awlen_o	output	AXI_LEN_W	AXI address write channel burst length.
axi_awsiz_o	output	3	AXI address write channel burst size.
axi_awburst_o	output	2	AXI address write channel burst type.
axi_awlock_o	output	2	AXI address write channel lock type.
axi_awcache_o	output	4	AXI address write channel memory type.
axi_awqos_o	output	4	AXI address write channel quality of service.
axi_wlast_o	output	1	AXI Write channel last word flag.
axi_bid_i	input	AXI_ID_W	AXI Write response channel ID.

Table 4: AXI manager interface for external memory

Name	Direction	Width	Description
inta_o	output	1	Interrupt Output A

Table 5: Interrupt Output

Name	Direction	Width	Description
phy_rstn_o	output	1	Reset signal for PHY. Duration configurable via PHY_RST_CNT parameter.

Table 6: PHY reset output (active low)

Name	Direction	Width	Description
mii_tx_clk_i	input	1	Transmit Nibble or Symbol Clock. The PHY provides the MTxCk signal. It operates at a frequency of 25 MHz (100 Mbps) or 2.5 MHz (10 Mbps). The clock is used as a timing reference for the transfer of MTxD[3:0], MtxEn, and MTxErr.
mii_txd_o	output	4	Transmit Data Nibble. Signals are the transmit data nibbles. They are synchronized to the rising edge of MTxCk. When MtxEn is asserted, PHY accepts the MTxD.
mii_tx_en_o	output	1	Transmit Enable. When asserted, this signal indicates to the PHY that the data MTxD[3:0] is valid and the transmission can start. The transmission starts with the first nibble of the preamble. The signal remains asserted until all nibbles to be transmitted are presented to the PHY. It is deasserted prior to the first MTxCk, following the final nibble of a frame.
mii_tx_er_o	output	1	Transmit Coding Error. When asserted for one MTxCk clock period while MtxEn is also asserted, this signal causes the PHY to transmit one or more symbols that are not part of the valid data or delimiter set somewhere in the frame being transmitted to indicate that there has been a transmit coding error.
mii_rx_clk_i	input	1	Receive Nibble or Symbol Clock. The PHY provides the MRxCk signal. It operates at a frequency of 25 MHz (100 Mbps) or 2.5 MHz (10 Mbps). The clock is used as a timing reference for the reception of MRxD[3:0], MRxDV, and MRxErr.
mii_rxd_i	input	4	Receive Data Nibble. These signals are the receive data nibble. They are synchronized to the rising edge of MRxCk. When MRxDV is asserted, the PHY sends a data nibble to the Rx MAC. For a correctly interpreted frame, seven bytes of a preamble and a completely formed SFD must be passed across the interface.
mii_rx_dv_i	input	1	Receive Data Valid. The PHY asserts this signal to indicate to the Rx MAC that it is presenting the valid nibbles on the MRxD[3:0] signals. The signal is asserted synchronously to the MRxCk. MRxDV is asserted from the first recovered nibble of the frame to the final recovered nibble. It is then deasserted prior to the first MRxCk that follows the final nibble.
mii_rx_er_i	input	1	Receive Error. The PHY asserts this signal to indicate to the Rx MAC that a media error was detected during the transmission of the current frame. MRxErr is synchronous to the MRxCk and is asserted for one or more MRxCk clock periods and then deasserted.
mii_crs_i	input	1	Carrier Sense. The PHY asynchronously asserts the carrier sense MCrS signal after the medium is detected in a non-idle state. When deasserted, this signal indicates that the media is in an idle state (and the transmission can start).
mii_col_i	input	1	Collision Detected. The PHY asynchronously asserts the collision signal MColl after the collision has been detected on the media. When deasserted, no collision is detected on the media.
mii_mdio_io	inout	1	Management Data Input/Output. Bi-directional serial data channel for PHY/STA communication.
mii_mdc_o	output	1	Management Data Clock. This is a clock for the MDIO serial data channel.

Table 7: MII interface



Name	Direction	Width	Description
csrs_axil_araddr_i	input	12	AXI-Lite address read channel byte address.
csrs_axil_arvalid_i	input	1	AXI-Lite address read channel valid.
csrs_axil_arready_o	output	1	AXI-Lite address read channel ready.
csrs_axil_rdata_o	output	DATA_W	AXI-Lite read channel data.
csrs_axil_rresp_o	output	2	AXI-Lite read channel response.
csrs_axil_rvalid_o	output	1	AXI-Lite read channel valid.
csrs_axil_rready_i	input	1	AXI-Lite read channel ready.
csrs_axil_awaddr_i	input	12	AXI-Lite address write channel byte address.
csrs_axil_awvalid_i	input	1	AXI-Lite address write channel valid.
csrs_axil_awready_o	output	1	AXI-Lite address write channel ready.
csrs_axil_wdata_i	input	DATA_W	AXI-Lite write channel data.
csrs_axil_wstrb_i	input	DATA_W/8	AXI-Lite write channel write strobe.
csrs_axil_wvalid_i	input	1	AXI-Lite write channel valid.
csrs_axil_wready_o	output	1	AXI-Lite write channel ready.
csrs_axil_bresp_o	output	2	AXI-Lite write response channel response.
csrs_axil_bvalid_o	output	1	AXI-Lite write response channel valid.
csrs_axil_bready_i	input	1	AXI-Lite write response channel ready.

Table 8: Control and status interface, when selecting the AXI-Lite CSR interface.

Name	Direction	Width	Description
csrs_wb_dat_o	output	DATA_W	Data input.
csrs_wb_datout_i	input	DATA_W	Data output.
csrs_wb_ack_o	output	1	Acknowledge input. Indicates normal termination of a bus cycle.
csrs_wb_adr_i	input	12	Address output. Passes binary address.
csrs_wb_cyc_i	input	1	Cycle output. Indicates a valid bus cycle.
csrs_wb_sel_i	input	DATA_W/8	Select output. Indicates where valid data is expected on the data bus.
csrs_wb_stb_i	input	1	Strobe output. Indicates valid access.
csrs_wb_we_i	input	1	Write enable. Indicates write access.

Table 9: Control and status interface, when selecting the Wishbone CSR interface.

Name	Direction	Width	Description
csrs_iob_valid_i	input	1	Request address is valid.
csrs_iob_addr_i	input	12	Byte address.
csrs_iob_wdata_i	input	DATA_W	Write data.
csrs_iob_wstrb_i	input	DATA_W/8	Write strobe.
csrs_iob_rvalid_o	output	1	Read data valid.
csrs_iob_rdata_o	output	DATA_W	Read data.
csrs_iob_ready_o	output	1	Interface ready.

Table 10: Control and Status Registers interface (auto-generated)

## 2.4 Control and Status Registers

The software accessible registers of the core are described in the following tables. The tables give information on the name, read/write capability, address, hardware and software width, and a textual description. The addresses are byte aligned and given in hexadecimal format. The hardware width is the number of bits that the register occupies in the hardware, while the software width is the number of bits that the register occupies in the software. In each address, the right-justified field having "Hw width" bits conveys the relevant information. Each register has only one type of access, either read or write, meaning that reading from a write-only register will produce invalid data or writing to a read-only register will not have any effect.

Name	R/W	Addr	Width		Default	Description
			Hw	Sw		
MODER	RW	0x0	32	32	40960	Mode Register
INT_SOURCE	RW	0x4	32	32	0	Interrupt Source Register
INT_MASK	RW	0x8	32	32	0	Interrupt Mask Register
IPGT	RW	0xC	32	32	18	Back to Back Inter Packet Gap Register
IPGR1	RW	0x10	32	32	12	Non Back to Back Inter Packet Gap Register 1
IPGR2	RW	0x14	32	32	18	Non Back to Back Inter Packet Gap Register 2
PACKETLEN	RW	0x18	32	32	4195840	Packet Length (minimum and maximum) Register
COLLCONF	RW	0x1C	32	32	61443	Collision and Retry Configuration
TX_BD_NUM	RW	0x20	32	32	64	Transmit Buffer Descriptor Number
CTRLMODER	RW	0x24	32	32	0	Control Module Mode Register
MIIMODER	RW	0x28	32	32	100	MII Mode Register
MIICOMMAND	RW	0x2C	32	32	0	MII Command Register
MIIADDRESS	RW	0x30	32	32	0	MII Address Register. Contains the PHY address and the register within the PHY address
MIITX_DATA	RW	0x34	32	32	0	MII Transmit Data. The data to be transmitted to the PHY
MIIRX_DATA	RW	0x38	32	32	0	MII Receive Data. The data received from the PHY
MIISTATUS	RW	0x3C	32	32	0	MII Status Register
MAC_ADDR0	RW	0x40	32	32	0	MAC Individual Address0. The LSB four bytes of the individual address are written to this register
MAC_ADDR1	RW	0x44	32	32	0	MAC Individual Address1. The MSB two bytes of the individual address are written to this register
ETH_HASH0_ADR	RW	0x48	32	32	0	HASH0 Register
ETH_HASH1_ADR	RW	0x4C	32	32	0	HASH1 Register
ETH_TXCTRL	RW	0x50	32	32	0	Transmit Control Register

Table 11: IOb\_Eth Software Accessible Registers

Name	R/W	Addr	Width		Default	Description
			Hw	Sw		
TX_BD_CNT	R	0x54	BD_NUM_LOG2	8	0	Buffer descriptor number of current TX frame
RX_BD_CNT	R	0x58	BD_NUM_LOG2	8	0	Buffer descriptor number of current RX frame
TX_WORD_CNT	R	0x5C	BUFFER_W	32	0	Word number of current TX frame
RX_WORD_CNT	R	0x60	BUFFER_W	32	0	Word number of current RX frame
RX_NBYTES	R	0x64	BUFFER_W	32	0	Size of received frame in bytes. Will be zero if no frame has been received. Will reset to zero when cpu completes reading the frame.
FRAME_WORD	RW	0x68	8	8	0	Frame word to transfer to/from buffer

Table 12: Data transfer/status registers for use without DMA

Name	R/W	Addr	Width		Default	Description
			Hw	Sw		
PHY_RST_VAL	R	0x6C	1	8	0	Current PHY reset signal value

Table 13: PHY reset control registers

Name	R/W	Addr	Width		Default	Description
			Hw	Sw		
BD	RW	0x400	32	32	0	Buffer descriptors

Table 14: IOb\_Eth Buffer Descriptors

Name	R/W	Addr	Width		Default	Description
			Hw	Sw		
VERSION	R	0x800	16	16	0001	Product version. This 16-bit register uses nibbles to represent decimal numbers using their binary values. The two most significant nibbles represent the integral part of the version, and the two least significant nibbles represent the decimal part. For example V12.34 is represented by 0x1234.

Table 15: General Registers.

## 3 Usage

### 3.1 Instantiation

Figure 3 illustrates how to instantiate the IP core and, if applicable, the required external subblocks.

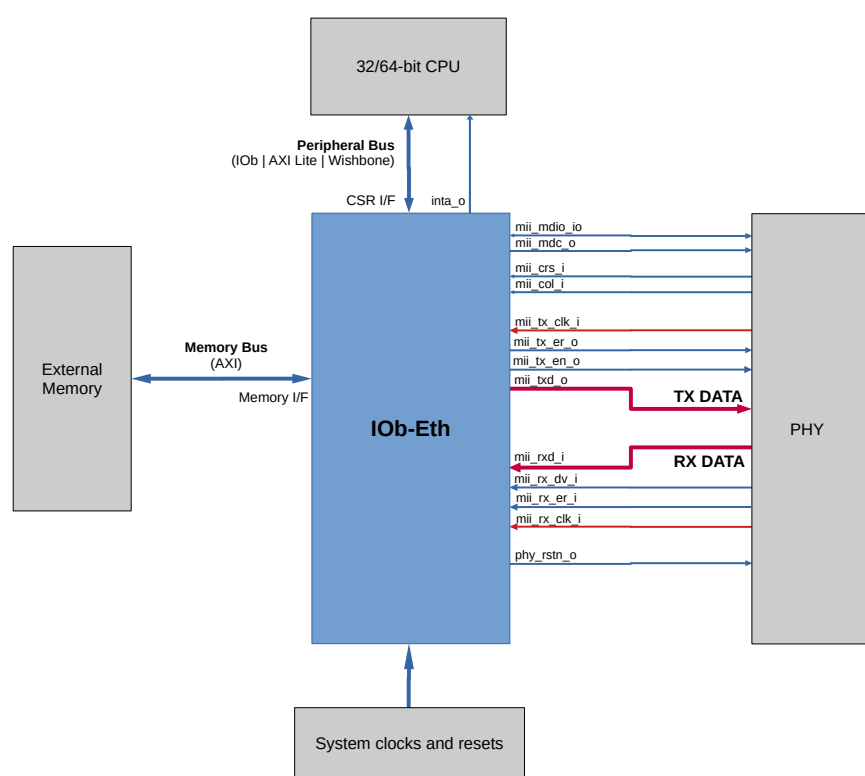


Figure 3: Core Instance and Required Surrounding Blocks

The CSRs bus (IOb native by default) should be connected to the desired manager component (e.g. a CPU), along with the ethernet interrupt output. The memory bus can optionally be connected to a memory controller. The MII interface and management signals connect to the PHY, along with the phy negated reset signal. The clock, clock enable, and reset ports can be connected to the desired clock and reset generator.

## 3.2 Simulation

The provided testbench uses the core instance described in Section 3.1. A high-level block diagram of the testbench is shown in Figure 4. The testbench is organized in a modular fashion, with each test described in a separate file. The test suite consists of all the test case files to make adding, modifying, or removing tests easy.

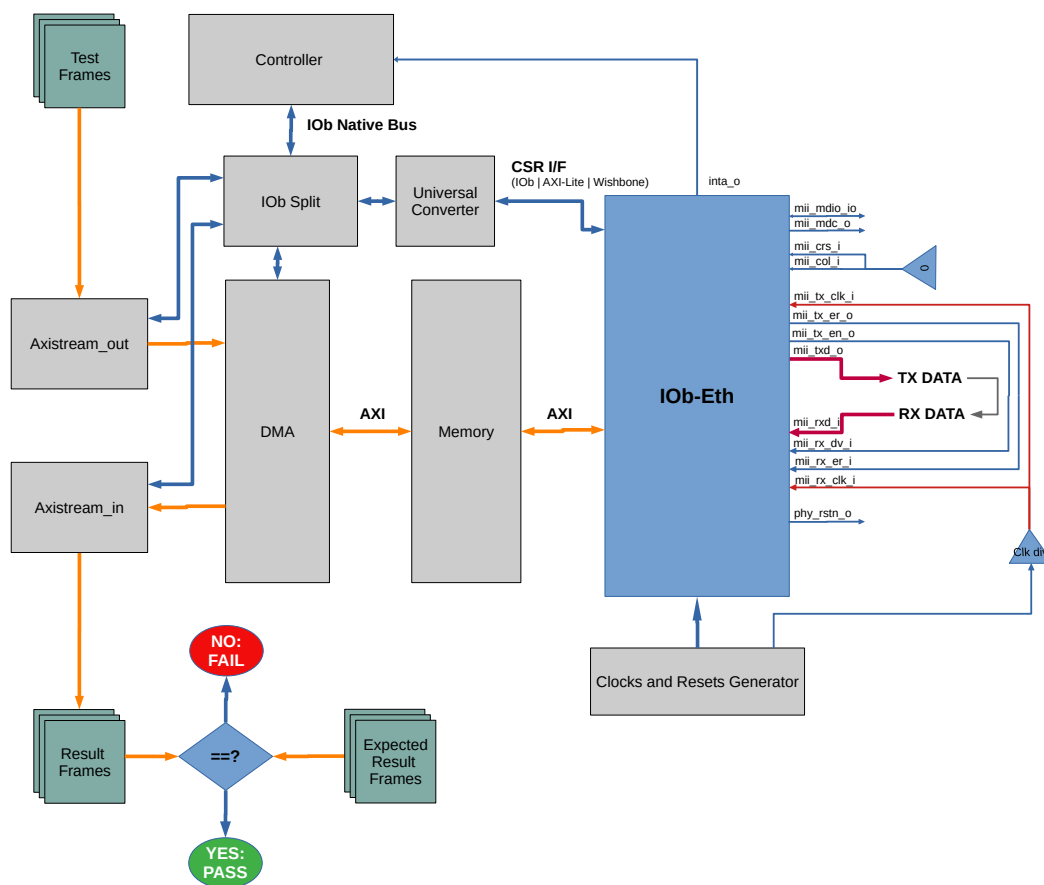


Figure 4: Testbench Block Diagram

The Ethernet testbench is configured to connect the IOb-Eth core's MII interface in loopback mode. The testbench architecture involves the following components and data flow:

- The IOb-Eth core's transmit and receive data is transferred to an external memory via its DMA/Memory interface.
- The external memory is populated with test frames using an axistream\_out peripheral and a DMA core.
- The received frames are stored in the memory and read out using the DMA core and an axistream\_in peripheral.

The testbench controller orchestrates the test sequence as follows:

1. Initializes all components.
2. Writes test frames to the axistream\_out peripheral via its IOB CSRs interface.
3. Commands the DMA core to transfer the test frames to the memory.
4. Commands the Ethernet core to transmit the test frames from the memory via its AXI memory interface.
5. Receives the loopback test frames into a new location in the memory.
6. Commands the DMA core to transfer the received frames from the memory to the axistream\_in peripheral.
7. Reads the frames from the axistream\_in peripheral and verifies that the data matches the original test frames sent.

## System-level Simulation

Upon request, simulation files to run the core embedded in a RISC-V system can be provided. The core is exercised in various modes by the RISC-V processor, using a bare-metal software program written in the C programming language.

## 4 Baremetal Drivers

### 4.1 eth\_frame\_struct.h File Reference

Defines related to the ethernet frame structure.

#### Macros

- #define **ETH\_TYPE\_H** 0x60  
*High byte of the Ethernet frame type.*
- #define **ETH\_TYPE\_L** 0x00  
*Low byte of the Ethernet frame type.*
- #define **ETH\_NBYTES** 1500  
*Maximum number of bytes in the Ethernet payload.*
- #define **ETH\_MINIMUM\_NBYTES** (64 - 18)  
*Minimum number of bytes in the Ethernet payload.*
- #define **HDR\_LEN** (2 \* IOB\_ETH\_MAC\_ADDR\_LEN + 2)  
*Length of the Ethernet header.*

#### Detailed Description

Defines related to the ethernet frame structure.

## 4.2 eth\_mem\_map.h File Reference

Memory map for the IOb-Eth core registers.

### Macros

- **#define ETH\_STATUS 0**  
*Offset for the status register.*
- **#define ETH\_SEND 1**  
*Offset for the send control register.*
- **#define ETH\_RCVACK 2**  
*Offset for the receive acknowledge register.*
- **#define ETH\_SOFTTRST 4**  
*Offset for the software reset register.*
- **#define ETH\_DUMMY 5**  
*Offset for the dummy register.*
- **#define ETH\_TX\_NBYTES 6**  
*Offset for the transmit number of bytes register.*
- **#define ETH\_RX\_NBYTES 7**  
*Offset for the receive number of bytes register.*
- **#define ETH\_CRC 8**  
*Offset for the CRC register.*
- **#define ETH\_RCV\_SIZE 9**  
*Offset for the receive size register.*
- **#define ETH\_DATA 2048**  
*Offset for the data buffer.*

### Detailed Description

Memory map for the IOb-Eth core registers.

## 4.3 iob\_eth.h File Reference

High-level driver functions for the IOb-Eth ethernet controller.

```
#include "stdint.h"  
#include <stdlib.h>
```

## Functions

- void **eth\_init** (int base\_address, void(\*clear\_cache\_func)(void))  
*Initializes the Ethernet core.*
- void **eth\_init\_clear\_cache** (void(\*clear\_cache\_func)(void))  
*Set the cache clearing function.*
- void **eth\_init\_mem\_alloc** (void \*(\*mem\_alloc\_func)(size\_t), void(\*mem\_free\_func)(void \*))  
*Set the memory allocation and free functions.*
- void **eth\_init\_mac** (int base\_address, uint64\_t mac\_addr, uint64\_t dest\_mac\_addr)  
*Initializes the MAC address and destination MAC address.*
- void **eth\_reset\_bd\_memory** ()  
*Resets the buffer descriptor memory.*
- unsigned short int **eth\_get\_payload\_size** (unsigned int idx)  
*Gets the payload size of a received frame.*
- void **eth\_set\_payload\_size** (unsigned int idx, unsigned int size)  
*Sets the payload size of a frame to be sent.*
- void **eth\_send\_frame** (char \*data\_to\_send, unsigned int size)  
*Sends an Ethernet frame.*
- int **eth\_prepare\_frame** (char \*external\_frame)  
*Prepare frame with ethernet template header.*
- void **eth\_send\_frame\_addr** (unsigned int size, uint32\_t frame\_addr)  
*Send an already prepared frame from a specific memory address.*
- int **eth\_check\_frame** (char \*data\_rcv, char \*frame\_ptr, unsigned int size)  
*Manual check for a valid frame at a specific memory address and copies its data.*
- int **eth\_rcv\_frame\_addr** (unsigned int size, int timeout, uint32\_t frame\_addr)  
*Receives a frame into a specific memory address.*
- int **eth\_rcv\_frame** (char \*data\_rcv, unsigned int size, int timeout)  
*Receives an Ethernet frame.*
- void **eth\_set\_receive\_timeout** (unsigned int timeout)  
*Sets the receive timeout value.*
- unsigned int **eth\_rcv\_file** (char \*data, int size)  
*Receives a file over Ethernet.*
- unsigned int **eth\_send\_file** (char \*data, int size)  
*Sends a file over Ethernet.*
- unsigned int **eth\_rcv\_variable\_file** (char \*data)  
*Receives a file of variable size.*
- unsigned int **eth\_send\_variable\_file** (char \*data, int size)  
*Sends a file of variable size.*
- void **eth\_wait\_phy\_rst** ()  
*Waits for the PHY to reset.*
- void **eth\_print\_status** ()  
*Prints the status of the Ethernet core.*

### 4.3.1 Detailed Description

High-level driver functions for the IOb-Eth ethernet controller.

This file provides the function declarations for interacting with the IOb-Eth hardware core.



## 4.3.2 Function Documentation

### eth\_check\_frame()

```
int eth_check_frame (  
    char * data_rcv,  
    char * frame_ptr,  
    unsigned int size)
```

Manual check for a valid frame at a specific memory address and copies its data.

#### Parameters

<i>data_rcv</i>	Buffer to store the received data.
<i>frame_ptr</i>	Pointer to the frame to check.
<i>size</i>	The expected size of the frame.

#### Returns

int 0 if a valid frame is found and copied, -1 otherwise.

### eth\_get\_payload\_size()

```
unsigned short int eth_get_payload_size (  
    unsigned int idx)
```

Gets the payload size of a received frame.

#### Parameters

<i>idx</i>	Index of the buffer descriptor.
------------	---------------------------------

#### Returns

unsigned short int The size of the payload.

### eth\_init()

```
void eth_init (  
    int base_address,  
    void(* clear_cache_func )(void))
```

Initializes the Ethernet core.

#### Parameters

<i>base_address</i>	The base memory address of the Ethernet core.
<i>clear_cache_func</i>	Function pointer to a function that clears the data cache.

### **eth\_init\_clear\_cache()**

```
void eth_init_clear_cache (  
    void(* clear_cache_func )(void))
```

Set the cache clearing function.

#### Parameters

<i>clear_cache_func</i>	Function pointer to a function that clears the data cache.
-------------------------	--

### **eth\_init\_mac()**

```
void eth_init_mac (  
    int base_address,  
    uint64_t mac_addr,  
    uint64_t dest_mac_addr)
```

Initializes the MAC address and destination MAC address.

#### Parameters

<i>base_address</i>	The base memory address of the Ethernet core.
<i>mac_addr</i>	The MAC address of the device.
<i>dest_mac_addr</i>	The destination MAC address.

### **eth\_init\_mem\_alloc()**

```
void eth_init_mem_alloc (  
    void *(* mem_alloc_func )(size_t),  
    void(* mem_free_func )(void *))
```

Set the memory allocation and free functions.

#### Parameters

<i>mem_alloc_func</i>	Function pointer for memory allocation.
<i>mem_free_func</i>	Function pointer for memory freeing.

## eth\_prepare\_frame()

```
int eth_prepare_frame (  
    char * external_frame)
```

Prepare frame with ethernet template header.

### Parameters

<i>external_frame</i>	should have at least TEMPLATE_LEN size.
-----------------------	---

### Returns

int 0 on success.

## eth\_rcv\_file()

```
unsigned int eth_rcv_file (  
    char * data,  
    int size)
```

Receives a file over Ethernet.

### Parameters

<i>data</i>	Buffer to store the received file data.
<i>size</i>	The size of the file to receive.

### Returns

unsigned int The number of bytes received.

## eth\_rcv\_frame()

```
int eth_rcv_frame (  
    char * data_rcv,  
    unsigned int size,  
    int timeout)
```

Receives an Ethernet frame.

### Parameters

<i>data_rcv</i>	Buffer to store the received data.
<i>size</i>	The number of bytes to be received.
<i>timeout</i>	The number of cycles (approximately) in which the data should be received.

#### Returns

int 0 if data is successfully received, -1 if a timeout occurs.

### **eth\_rcv\_frame\_addr()**

```
int eth_rcv_frame_addr (  
    unsigned int size,  
    int timeout,  
    uint32_t frame_addr)
```

Receives a frame into a specific memory address.

#### Parameters

<i>size</i>	The expected size of the frame.
<i>timeout</i>	The number of cycles to wait for the frame.
<i>frame_addr</i>	The memory address to store the received frame.

#### Returns

int 0 if the frame is successfully received, -1 on timeout.

### **eth\_rcv\_variable\_file()**

```
unsigned int eth_rcv_variable_file (  
    char * data)
```

Receives a file of variable size.

#### Parameters

<i>data</i>	Buffer to store the received file data.
-------------	---

#### Returns

unsigned int The size of the received file.

### **eth\_send\_file()**

```
unsigned int eth_send_file (  
    char * data,  
    int size)
```

Sends a file over Ethernet.

**Parameters**

<i>data</i>	Pointer to the file data to send.
<i>size</i>	The size of the file to send.

**Returns**

unsigned int The number of bytes sent.

**eth\_send\_frame()**

```
void eth_send_frame (  
    char * data_to_send,  
    unsigned int size)
```

Sends an Ethernet frame.

**Warning**

Care when using this function directly, too small a size or too large might not work (frame does not get sent)

**Parameters**

<i>data_to_send</i>	Pointer to the data to be sent.
<i>size</i>	Size of the data to be sent.

**eth\_send\_frame\_addr()**

```
void eth_send_frame_addr (  
    unsigned int size,  
    uint32_t frame_addr)
```

Send an already prepared frame from a specific memory address.

**Parameters**

<i>size</i>	The size of the frame to send.
<i>frame_addr</i>	The memory address of the frame.

## **eth\_send\_variable\_file()**

```
unsigned int eth_send_variable_file (  
    char * data,  
    int size)
```

Sends a file of variable size.

### **Parameters**

<i>data</i>	Pointer to the file data to send.
<i>size</i>	The size of the file to send.

### **Returns**

unsigned int The number of bytes sent.

## **eth\_set\_payload\_size()**

```
void eth_set_payload_size (  
    unsigned int idx,  
    unsigned int size)
```

Sets the payload size of a frame to be sent.

### **Parameters**

<i>idx</i>	Index of the buffer descriptor.
<i>size</i>	The size of the payload.

## **eth\_set\_receive\_timeout()**

```
void eth_set_receive_timeout (  
    unsigned int timeout)
```

Sets the receive timeout value.

### **Parameters**

<i>timeout</i>	The timeout value in cycles.
----------------	------------------------------

## 4.4 iob\_eth\_csrs.h File Reference

Function prototypes for the iob\_eth core.

```
#include <stdint.h>
#include "iob_eth_csrs_conf.h"
```

### Functions

- void iob\_eth\_csrs\_init\_baseaddr (uint32\_t addr)  
*Set core base address.*
- void iob\_write (uint32\_t addr, uint32\_t data\_w, uint32\_t value)  
*Write access function prototype.*
- uint32\_t iob\_read (uint32\_t addr, uint32\_t data\_w)  
*Read access function prototype.*
- void iob\_eth\_csrs\_set\_moder (uint32\_t value)  
*Set moder value. Mode Register.*
- uint32\_t iob\_eth\_csrs\_get\_moder ()  
*Get moder value. Mode Register.*
- void iob\_eth\_csrs\_set\_int\_source (uint32\_t value)  
*Set int\_source value. Interrupt Source Register.*
- uint32\_t iob\_eth\_csrs\_get\_int\_source ()  
*Get int\_source value. Interrupt Source Register.*
- void iob\_eth\_csrs\_set\_int\_mask (uint32\_t value)  
*Set int\_mask value. Interrupt Mask Register.*
- uint32\_t iob\_eth\_csrs\_get\_int\_mask ()  
*Get int\_mask value. Interrupt Mask Register.*
- void iob\_eth\_csrs\_set\_ipgt (uint32\_t value)  
*Set ipgt value. Back to Back Inter Packet Gap Register.*
- uint32\_t iob\_eth\_csrs\_get\_ipgt ()  
*Get ipgt value. Back to Back Inter Packet Gap Register.*
- void iob\_eth\_csrs\_set\_ipgr1 (uint32\_t value)  
*Set ipgr1 value. Non Back to Back Inter Packet Gap Register 1.*
- uint32\_t iob\_eth\_csrs\_get\_ipgr1 ()  
*Get ipgr1 value. Non Back to Back Inter Packet Gap Register 1.*
- void iob\_eth\_csrs\_set\_ipgr2 (uint32\_t value)  
*Set ipgr2 value. Non Back to Back Inter Packet Gap Register 2.*
- uint32\_t iob\_eth\_csrs\_get\_ipgr2 ()  
*Get ipgr2 value. Non Back to Back Inter Packet Gap Register 2.*
- void iob\_eth\_csrs\_set\_packetlen (uint32\_t value)  
*Set packetlen value. Packet Length (minimum and maximum) Register.*
- uint32\_t iob\_eth\_csrs\_get\_packetlen ()  
*Get packetlen value. Packet Length (minimum and maximum) Register.*
- void iob\_eth\_csrs\_set\_collconf (uint32\_t value)  
*Set collconf value. Collision and Retry Configuration.*
- uint32\_t iob\_eth\_csrs\_get\_collconf ()

- Get collconf value. Collision and Retry Configuration.*

  - void iob\_eth\_csrs\_set\_tx\_bd\_num (uint32\_t value)
 

*Set tx\_bd\_num value. Transmit Buffer Descriptor Number.*
  - uint32\_t iob\_eth\_csrs\_get\_tx\_bd\_num ()
 

*Get tx\_bd\_num value. Transmit Buffer Descriptor Number.*
  - void iob\_eth\_csrs\_set\_ctrlmoder (uint32\_t value)
 

*Set ctrlmoder value. Control Module Mode Register.*
  - uint32\_t iob\_eth\_csrs\_get\_ctrlmoder ()
 

*Get ctrlmoder value. Control Module Mode Register.*
  - void iob\_eth\_csrs\_set\_miimoder (uint32\_t value)
 

*Set miimoder value. MII Mode Register.*
  - uint32\_t iob\_eth\_csrs\_get\_miimoder ()
 

*Get miimoder value. MII Mode Register.*
  - void iob\_eth\_csrs\_set\_miicommand (uint32\_t value)
 

*Set miicommand value. MII Command Register.*
  - uint32\_t iob\_eth\_csrs\_get\_miicommand ()
 

*Get miicommand value. MII Command Register.*
  - void iob\_eth\_csrs\_set\_miiaddress (uint32\_t value)
 

*Set miiaddress value. MII Address Register. Contains the PHY address and the register within the PHY address.*
  - uint32\_t iob\_eth\_csrs\_get\_miiaddress ()
 

*Get miiaddress value. MII Address Register. Contains the PHY address and the register within the PHY address.*
  - void iob\_eth\_csrs\_set\_mii\_tx\_data (uint32\_t value)
 

*Set mii\_tx\_data value. MII Transmit Data. The data to be transmitted to the PHY.*
  - uint32\_t iob\_eth\_csrs\_get\_mii\_tx\_data ()
 

*Get mii\_tx\_data value. MII Transmit Data. The data to be transmitted to the PHY.*
  - void iob\_eth\_csrs\_set\_mii\_rx\_data (uint32\_t value)
 

*Set mii\_rx\_data value. MII Receive Data. The data received from the PHY.*
  - uint32\_t iob\_eth\_csrs\_get\_mii\_rx\_data ()
 

*Get mii\_rx\_data value. MII Receive Data. The data received from the PHY.*
  - void iob\_eth\_csrs\_set\_miistatus (uint32\_t value)
 

*Set miistatus value. MII Status Register.*
  - uint32\_t iob\_eth\_csrs\_get\_miistatus ()
 

*Get miistatus value. MII Status Register.*
  - void iob\_eth\_csrs\_set\_mac\_addr0 (uint32\_t value)
 

*Set mac\_addr0 value. MAC Individual Address0. The LSB four bytes of the individual address are written to this register.*
  - uint32\_t iob\_eth\_csrs\_get\_mac\_addr0 ()
 

*Get mac\_addr0 value. MAC Individual Address0. The LSB four bytes of the individual address are written to this register.*
  - void iob\_eth\_csrs\_set\_mac\_addr1 (uint32\_t value)
 

*Set mac\_addr1 value. MAC Individual Address1. The MSB two bytes of the individual address are written to this register.*
  - uint32\_t iob\_eth\_csrs\_get\_mac\_addr1 ()
 

*Get mac\_addr1 value. MAC Individual Address1. The MSB two bytes of the individual address are written to this register.*
  - void iob\_eth\_csrs\_set\_eth\_hash0\_adr (uint32\_t value)
 

*Set eth\_hash0\_adr value. HASH0 Register.*
  - uint32\_t iob\_eth\_csrs\_get\_eth\_hash0\_adr ()



- *Get eth\_hash0\_adr value. HASH0 Register.*  
 • void iob\_eth\_csrs\_set\_eth\_hash1\_adr (uint32\_t value)  
   *Set eth\_hash1\_adr value. HASH1 Register.*  
 • uint32\_t iob\_eth\_csrs\_get\_eth\_hash1\_adr ()  
   *Get eth\_hash1\_adr value. HASH1 Register.*  
 • void iob\_eth\_csrs\_set\_eth\_txctrl (uint32\_t value)  
   *Set eth\_txctrl value. Transmit Control Register.*  
 • uint32\_t iob\_eth\_csrs\_get\_eth\_txctrl ()  
   *Get eth\_txctrl value. Transmit Control Register.*  
 • uint8\_t iob\_eth\_csrs\_get\_tx\_bd\_cnt ()  
   *Get tx\_bd\_cnt value. Buffer descriptor number of current TX frame.*  
 • uint8\_t iob\_eth\_csrs\_get\_rx\_bd\_cnt ()  
   *Get rx\_bd\_cnt value. Buffer descriptor number of current RX frame.*  
 • uint32\_t iob\_eth\_csrs\_get\_tx\_word\_cnt ()  
   *Get tx\_word\_cnt value. Word number of current TX frame.*  
 • uint32\_t iob\_eth\_csrs\_get\_rx\_word\_cnt ()  
   *Get rx\_word\_cnt value. Word number of current RX frame.*  
 • uint32\_t iob\_eth\_csrs\_get\_rx\_nbytes ()  
   *Get rx\_nbytes value. Size of received frame in bytes. Will be zero if no frame has been received. Will reset to zero when cpu completes reading the frame.*  
 • void iob\_eth\_csrs\_set\_frame\_word (uint8\_t value)  
   *Set frame\_word value. Frame word to transfer to/from buffer.*  
 • uint8\_t iob\_eth\_csrs\_get\_frame\_word ()  
   *Get frame\_word value. Frame word to transfer to/from buffer.*  
 • uint8\_t iob\_eth\_csrs\_get\_phy\_rst\_val ()  
   *Get phy\_rst\_val value. Current PHY reset signal value.*  
 • void iob\_eth\_csrs\_set\_bd (uint32\_t value, int addr)  
   *Set bd value. Buffer descriptors.*  
 • uint32\_t iob\_eth\_csrs\_get\_bd (int addr)  
   *Get bd value. Buffer descriptors.*  
 • uint16\_t iob\_eth\_csrs\_get\_version ()  
   *Get version value. Product version. This 16-bit register uses nibbles to represent decimal numbers using their binary values. The two most significant nibbles represent the integral part of the version, and the two least significant nibbles represent the decimal part. For example V12.34 is represented by 0x1234.*

### 4.4.1 Detailed Description

Function prototypes for the iob\_eth core.

This file contains the function prototypes to access the Control and Status Registers (CSRs) for the iob\_eth core.

This file is automatically generated by Py2HWSW

## 4.4.2 Function Documentation

### **iob\_eth\_csrs\_get\_bd()**

```
uint32_t iob_eth_csrs_get_bd (  
    int addr)
```

Get bd value. Buffer descriptors.

#### Parameters

<i>addr</i>	bd array address.
-------------	-------------------

#### Returns

uint32\_t bd value.

### **iob\_eth\_csrs\_get\_collconf()**

```
uint32_t iob_eth_csrs_get_collconf ()
```

Get collconf value. Collision and Retry Configuration.

#### Returns

uint32\_t collconf value.

### **iob\_eth\_csrs\_get\_ctrlmoder()**

```
uint32_t iob_eth_csrs_get_ctrlmoder ()
```

Get ctrlmoder value. Control Module Mode Register.

#### Returns

uint32\_t ctrlmoder value.

### **iob\_eth\_csrs\_get\_eth\_hash0\_adr()**

```
uint32_t iob_eth_csrs_get_eth_hash0_adr ()
```

Get eth\_hash0\_adr value. HASH0 Register.

#### Returns

uint32\_t eth\_hash0\_adr value.

**iob\_eth\_csrs\_get\_eth\_hash1\_adr()**

```
uint32_t iob_eth_csrs_get_eth_hash1_adr ()
```

Get eth\_hash1\_adr value. HASH1 Register.

**Returns**

uint32\_t eth\_hash1\_adr value.

**iob\_eth\_csrs\_get\_eth\_txctrl()**

```
uint32_t iob_eth_csrs_get_eth_txctrl ()
```

Get eth\_txctrl value. Transmit Control Register.

**Returns**

uint32\_t eth\_txctrl value.

**iob\_eth\_csrs\_get\_frame\_word()**

```
uint8_t iob_eth_csrs_get_frame_word ()
```

Get frame\_word value. Frame word to transfer to/from buffer.

**Returns**

uint8\_t frame\_word value.

**iob\_eth\_csrs\_get\_int\_mask()**

```
uint32_t iob_eth_csrs_get_int_mask ()
```

Get int\_mask value. Interrupt Mask Register.

**Returns**

uint32\_t int\_mask value.

**iob\_eth\_csrs\_get\_int\_source()**

```
uint32_t iob_eth_csrs_get_int_source ()
```

Get int\_source value. Interrupt Source Register.

Returns

uint32\_t int\_source value.

### **iob\_eth\_csrs\_get\_ipgr1()**

uint32\_t iob\_eth\_csrs\_get\_ipgr1 ()

Get ipgr1 value. Non Back to Back Inter Packet Gap Register 1.

Returns

uint32\_t ipgr1 value.

### **iob\_eth\_csrs\_get\_ipgr2()**

uint32\_t iob\_eth\_csrs\_get\_ipgr2 ()

Get ipgr2 value. Non Back to Back Inter Packet Gap Register 2.

Returns

uint32\_t ipgr2 value.

### **iob\_eth\_csrs\_get\_ipgt()**

uint32\_t iob\_eth\_csrs\_get\_ipgt ()

Get ipgt value. Back to Back Inter Packet Gap Register.

Returns

uint32\_t ipgt value.

### **iob\_eth\_csrs\_get\_mac\_addr0()**

uint32\_t iob\_eth\_csrs\_get\_mac\_addr0 ()

Get mac\_addr0 value. MAC Individual Address0. The LSB four bytes of the individual address are written to this register.

Returns

uint32\_t mac\_addr0 value.

**iob\_eth\_csrs\_get\_mac\_addr1()**

```
uint32_t iob_eth_csrs_get_mac_addr1 ()
```

Get mac\_addr1 value. MAC Individual Address1. The MSB two bytes of the individual address are written to this register.

**Returns**

uint32\_t mac\_addr1 value.

**iob\_eth\_csrs\_get\_miiaddress()**

```
uint32_t iob_eth_csrs_get_miiaddress ()
```

Get miiaddress value. MII Address Register. Contains the PHY address and the register within the PHY address.

**Returns**

uint32\_t miiaddress value.

**iob\_eth\_csrs\_get\_miiccommand()**

```
uint32_t iob_eth_csrs_get_miiccommand ()
```

Get miiccommand value. MII Command Register.

**Returns**

uint32\_t miiccommand value.

**iob\_eth\_csrs\_get\_miimoder()**

```
uint32_t iob_eth_csrs_get_miimoder ()
```

Get miimoder value. MII Mode Register.

**Returns**

uint32\_t miimoder value.

**iob\_eth\_csrs\_get\_miirx\_data()**

```
uint32_t iob_eth_csrs_get_miirx_data ()
```

Get miirx\_data value. MII Receive Data. The data received from the PHY.

Returns

uint32\_t miirx\_data value.

### **iob\_eth\_csrs\_get\_miistatus()**

```
uint32_t iob_eth_csrs_get_miistatus ()
```

Get miistatus value. MII Status Register.

Returns

uint32\_t miistatus value.

### **iob\_eth\_csrs\_get\_miitx\_data()**

```
uint32_t iob_eth_csrs_get_miitx_data ()
```

Get miitx\_data value. MII Transmit Data. The data to be transmitted to the PHY.

Returns

uint32\_t miitx\_data value.

### **iob\_eth\_csrs\_get\_moder()**

```
uint32_t iob_eth_csrs_get_moder ()
```

Get moder value. Mode Register.

Returns

uint32\_t moder value.

### **iob\_eth\_csrs\_get\_packetlen()**

```
uint32_t iob_eth_csrs_get_packetlen ()
```

Get packetlen value. Packet Length (minimum and maximum) Register.

Returns

uint32\_t packetlen value.

**iob\_eth\_csrs\_get\_phy\_rst\_val()**

```
uint8_t iob_eth_csrs_get_phy_rst_val ()
```

Get phy\_rst\_val value. Current PHY reset signal value.

Returns

uint8\_t phy\_rst\_val value.

**iob\_eth\_csrs\_get\_rx\_bd\_cnt()**

```
uint8_t iob_eth_csrs_get_rx_bd_cnt ()
```

Get rx\_bd\_cnt value. Buffer descriptor number of current RX frame.

Returns

uint8\_t rx\_bd\_cnt value.

**iob\_eth\_csrs\_get\_rx\_nbytes()**

```
uint32_t iob_eth_csrs_get_rx_nbytes ()
```

Get rx\_nbytes value. Size of received frame in bytes. Will be zero if no frame has been received. Will reset to zero when cpu completes reading the frame.

Returns

uint32\_t rx\_nbytes value.

**iob\_eth\_csrs\_get\_rx\_word\_cnt()**

```
uint32_t iob_eth_csrs_get_rx_word_cnt ()
```

Get rx\_word\_cnt value. Word number of current RX frame.

Returns

uint32\_t rx\_word\_cnt value.

**iob\_eth\_csrs\_get\_tx\_bd\_cnt()**

```
uint8_t iob_eth_csrs_get_tx_bd_cnt ()
```

Get tx\_bd\_cnt value. Buffer descriptor number of current TX frame.

Returns

uint8\_t tx\_bd\_cnt value.

### **iob\_eth\_csrs\_get\_tx\_bd\_num()**

```
uint32_t iob_eth_csrs_get_tx_bd_num ()
```

Get tx\_bd\_num value. Transmit Buffer Descriptor Number.

Returns

uint32\_t tx\_bd\_num value.

### **iob\_eth\_csrs\_get\_tx\_word\_cnt()**

```
uint32_t iob_eth_csrs_get_tx_word_cnt ()
```

Get tx\_word\_cnt value. Word number of current TX frame.

Returns

uint32\_t tx\_word\_cnt value.

### **iob\_eth\_csrs\_get\_version()**

```
uint16_t iob_eth_csrs_get_version ()
```

Get version value. Product version. This 16-bit register uses nibbles to represent decimal numbers using their binary values. The two most significant nibbles represent the integral part of the version, and the two least significant nibbles represent the decimal part. For example V12.34 is represented by 0x1234.

Returns

uint16\_t version value.

### **iob\_eth\_csrs\_init\_baseaddr()**

```
void iob_eth_csrs_init_baseaddr (  
    uint32_t addr)
```

Set core base address.

This function sets the base address for the core in the system. All other accesses are offset from this base address.



**Parameters**

<i>addr</i>	Base address for core.
-------------	------------------------

**iob\_eth\_csrs\_set\_bd()**

```
void iob_eth_csrs_set_bd (  
    uint32_t value,  
    int addr)
```

Set bd value. Buffer descriptors.

**Parameters**

<i>value</i>	bd Value.
<i>addr</i>	bd array address.

**iob\_eth\_csrs\_set\_collconf()**

```
void iob_eth_csrs_set_collconf (  
    uint32_t value)
```

Set collconf value. Collision and Retry Configuration.

**Parameters**

<i>value</i>	collconf Value.
--------------	-----------------

**iob\_eth\_csrs\_set\_ctrlmoder()**

```
void iob_eth_csrs_set_ctrlmoder (  
    uint32_t value)
```

Set ctrlmoder value. Control Module Mode Register.

**Parameters**

<i>value</i>	ctrlmoder Value.
--------------	------------------

### **iob\_eth\_csrs\_set\_eth\_hash0\_adr()**

```
void iob_eth_csrs_set_eth_hash0_adr (  
    uint32_t value)
```

Set eth\_hash0\_adr value. HASH0 Register.

#### **Parameters**

<i>value</i>	eth_hash0_adr Value.
--------------	----------------------

### **iob\_eth\_csrs\_set\_eth\_hash1\_adr()**

```
void iob_eth_csrs_set_eth_hash1_adr (  
    uint32_t value)
```

Set eth\_hash1\_adr value. HASH1 Register.

#### **Parameters**

<i>value</i>	eth_hash1_adr Value.
--------------	----------------------

### **iob\_eth\_csrs\_set\_eth\_txctrl()**

```
void iob_eth_csrs_set_eth_txctrl (  
    uint32_t value)
```

Set eth\_txctrl value. Transmit Control Register.

#### **Parameters**

<i>value</i>	eth_txctrl Value.
--------------	-------------------

### **iob\_eth\_csrs\_set\_frame\_word()**

```
void iob_eth_csrs_set_frame_word (  
    uint8_t value)
```

Set frame\_word value. Frame word to transfer to/from buffer.

#### **Parameters**

<i>value</i>	frame_word Value.
--------------	-------------------

### **iob\_eth\_csrs\_set\_int\_mask()**

```
void iob_eth_csrs_set_int_mask (  
    uint32_t value)
```

Set int\_mask value. Interrupt Mask Register.

#### **Parameters**

<i>value</i>	int_mask Value.
--------------	-----------------

### **iob\_eth\_csrs\_set\_int\_source()**

```
void iob_eth_csrs_set_int_source (  
    uint32_t value)
```

Set int\_source value. Interrupt Source Register.

#### **Parameters**

<i>value</i>	int_source Value.
--------------	-------------------

### **iob\_eth\_csrs\_set\_ipgr1()**

```
void iob_eth_csrs_set_ipgr1 (  
    uint32_t value)
```

Set ipgr1 value. Non Back to Back Inter Packet Gap Register 1.

#### **Parameters**

<i>value</i>	ipgr1 Value.
--------------	--------------

### **iob\_eth\_csrs\_set\_ipgr2()**

```
void iob_eth_csrs_set_ipgr2 (  
    uint32_t value)
```

Set ipgr2 value. Non Back to Back Inter Packet Gap Register 2.

#### **Parameters**

<i>value</i>	ipgr2 Value.
--------------	--------------

### **iob\_eth\_csrs\_set\_ipgt()**

```
void iob_eth_csrs_set_ipgt (  
    uint32_t value)
```

Set ipgt value. Back to Back Inter Packet Gap Register.

#### **Parameters**

<i>value</i>	ipgt Value.
--------------	-------------

### **iob\_eth\_csrs\_set\_mac\_addr0()**

```
void iob_eth_csrs_set_mac_addr0 (  
    uint32_t value)
```

Set mac\_addr0 value. MAC Individual Address0. The LSB four bytes of the individual address are written to this register.

#### **Parameters**

<i>value</i>	mac_addr0 Value.
--------------	------------------

### **iob\_eth\_csrs\_set\_mac\_addr1()**

```
void iob_eth_csrs_set_mac_addr1 (  
    uint32_t value)
```

Set mac\_addr1 value. MAC Individual Address1. The MSB two bytes of the individual address are written to this register.

#### **Parameters**

<i>value</i>	mac_addr1 Value.
--------------	------------------

### **iob\_eth\_csrs\_set\_miiaddress()**

```
void iob_eth_csrs_set_miiaddress (  
    uint32_t value)
```

Set miiaddress value. MII Address Register. Contains the PHY address and the register within the PHY address.

**Parameters**

<i>value</i>	miiaddress Value.
--------------	-------------------

**iob\_eth\_csrs\_set\_miicommand()**

```
void iob_eth_csrs_set_miicommand (  
    uint32_t value)
```

Set miicommand value. MII Command Register.

**Parameters**

<i>value</i>	miicommand Value.
--------------	-------------------

**iob\_eth\_csrs\_set\_miimoder()**

```
void iob_eth_csrs_set_miimoder (  
    uint32_t value)
```

Set miimoder value. MII Mode Register.

**Parameters**

<i>value</i>	miimoder Value.
--------------	-----------------

**iob\_eth\_csrs\_set\_miirx\_data()**

```
void iob_eth_csrs_set_miirx_data (  
    uint32_t value)
```

Set miirx\_data value. MII Receive Data. The data received from the PHY.

**Parameters**

<i>value</i>	miirx_data Value.
--------------	-------------------

**iob\_eth\_csrs\_set\_miistatus()**

```
void iob_eth_csrs_set_miistatus (  
    uint32_t value)
```

Set miistatus value. MII Status Register.

**Parameters**

<i>value</i>	miistatus Value.
--------------	------------------

**iob\_eth\_csrs\_set\_miitx\_data()**

```
void iob_eth_csrs_set_miitx_data (  
    uint32_t value)
```

Set miitx\_data value. MII Transmit Data. The data to be transmitted to the PHY.

**Parameters**

<i>value</i>	miitx_data Value.
--------------	-------------------

**iob\_eth\_csrs\_set\_moder()**

```
void iob_eth_csrs_set_moder (  
    uint32_t value)
```

Set moder value. Mode Register.

**Parameters**

<i>value</i>	moder Value.
--------------	--------------

**iob\_eth\_csrs\_set\_packetlen()**

```
void iob_eth_csrs_set_packetlen (  
    uint32_t value)
```

Set packetlen value. Packet Length (minimum and maximum) Register.

**Parameters**

<i>value</i>	packetlen Value.
--------------	------------------

## **iob\_eth\_csrs\_set\_tx\_bd\_num()**

```
void iob_eth_csrs_set_tx_bd_num (  
    uint32_t value)
```

Set tx\_bd\_num value. Transmit Buffer Descriptor Number.

### **Parameters**

<i>value</i>	tx_bd_num Value.
--------------	------------------

## **iob\_read()**

```
uint32_t iob_read (  
    uint32_t addr,  
    uint32_t data_w)
```

Read access function prototype.

### **Parameters**

<i>addr</i>	Address to write to.
<i>data_w</i>	Data width in bits.

### **Returns**

uint32\_t Read data value.

## **iob\_write()**

```
void iob_write (  
    uint32_t addr,  
    uint32_t data_w,  
    uint32_t value)
```

Write access function prototype.

### **Parameters**

<i>addr</i>	Address to write to.
<i>data_w</i>	Data width in bits.
<i>value</i>	Value to write.

## 4.5 iob\_eth\_defines.h File Reference

Macros for controlling the IOb-Eth core.

```
#include "eth_frame_struct.h"
#include "iob_eth_conf.h"
#include "iob_eth_csrs.h"
#include "iob_eth_macros.h"
```

### Macros

- `#define eth_tx_ready(idx)`  
*Check if the transmit buffer descriptor is ready.*
- `#define eth_rx_ready(idx)`  
*Check if the receive buffer descriptor is ready.*
- `#define eth_bad_crc(idx)`  
*Check if the received frame has a bad CRC.*
- `#define eth_send(enable)`  
*Enable or disable the transmitter.*
- `#define eth_receive(enable)`  
*Enable or disable the receiver.*
- `#define eth_set_ready(idx, enable)`  
*Set the ready flag for a transmit buffer descriptor.*
- `#define eth_set_empty(idx, enable)`  
*Set the empty flag for a receive buffer descriptor.*
- `#define eth_set_interrupt(idx, enable)`  
*Enable or disable interrupts for a buffer descriptor.*
- `#define eth_set_wr(idx, enable)`  
*Set the wrap flag for a buffer descriptor.*
- `#define eth_set_crc(idx, enable)`  
*Enable or disable CRC generation for a transmit buffer descriptor.*
- `#define eth_set_pad(idx, enable)`  
*Enable or disable padding for a transmit buffer descriptor.*
- `#define eth_set_ptr(idx, ptr)`  
*Set the data pointer for a buffer descriptor.*

### 4.5.1 Detailed Description

Macros for controlling the IOb-Eth core.



## 4.5.2 Macro Definition Documentation

### eth\_bad\_crc

```
#define eth_bad_crc(  
    idx)
```

#### Value:

```
((iob_eth_csrs_get_bd(idx < 1) & RX_BD_CRC) || 0)
```

Check if the received frame has a bad CRC.

#### Parameters

<i>idx</i>	Index of the buffer descriptor.
------------	---------------------------------

#### Returns

1 if CRC is bad, 0 otherwise.

### eth\_receive

```
#define eth_receive(  
    enable)
```

#### Value:

```
{  
    iob_eth_csrs_set_moder(iob_eth_csrs_get_moder() & ~MODER_RXEN |  
                           (enable ? MODER_RXEN : 0));  
}
```

Enable or disable the receiver.

#### Parameters

<i>enable</i>	1 to enable, 0 to disable.
---------------	----------------------------

### eth\_rx\_ready

```
#define eth_rx_ready(  
    idx)
```

#### Value:

```
eth_tx_ready(idx)
```

Check if the receive buffer descriptor is ready.

**Parameters**

<i>idx</i>	Index of the buffer descriptor.
------------	---------------------------------

**Returns**

1 if ready, 0 otherwise.

**eth\_send**

```
#define eth_send(  
    enable)
```

**Value:**

```
{  
    iob_eth_csrs_set_moder(iob_eth_csrs_get_moder() & ~MODER_TXEN |  
                           (enable ? MODER_TXEN : 0));  
}
```

Enable or disable the transmitter.

**Parameters**

<i>enable</i>	1 to enable, 0 to disable.
---------------	----------------------------

**eth\_set\_crc**

```
#define eth_set_crc(  
    idx,  
    enable)
```

**Value:**

```
{  
    iob_eth_csrs_set_bd(iob_eth_csrs_get_bd(idx << 1) & ~TX_BD_CRC |  
                        (enable ? TX_BD_CRC : 0),  
                        idx << 1);  
}
```

Enable or disable CRC generation for a transmit buffer descriptor.

**Parameters**

<i>idx</i>	Index of the buffer descriptor.
<i>enable</i>	1 to enable CRC, 0 to disable.

## eth\_set\_empty

```
#define eth_set_empty(
    idx,
    enable)
```

### Value:

```
eth_set_ready(idx, enable)
```

Set the empty flag for a receive buffer descriptor.

### Parameters

<i>idx</i>	Index of the buffer descriptor.
<i>enable</i>	1 to set as empty, 0 otherwise.

## eth\_set\_interrupt

```
#define eth_set_interrupt(
    idx,
    enable)
```

### Value:

```
{
    iob_eth_csrs_set_bd(iob_eth_csrs_get_bd(idx < 1) & ~TX_BD_IRQ |
        (enable ? TX_BD_IRQ : 0),
        idx < 1);
}
```

Enable or disable interrupts for a buffer descriptor.

### Parameters

<i>idx</i>	Index of the buffer descriptor.
<i>enable</i>	1 to enable, 0 to disable.

## eth\_set\_pad

```
#define eth_set_pad(
    idx,
    enable)
```

### Value:

```
{
    iob_eth_csrs_set_bd(iob_eth_csrs_get_bd(idx < 1) & ~TX_BD_PAD |
        (enable ? TX_BD_PAD : 0),
        idx < 1);
}
```

Enable or disable padding for a transmit buffer descriptor.

#### Parameters

<i>idx</i>	Index of the buffer descriptor.
<i>enable</i>	1 to enable padding, 0 to disable.

### eth\_set\_ptr

```
#define eth_set_ptr(  
    idx,  
    ptr)
```

#### Value:

```
{ iob_eth_csrs_set_bd((uint32_t)(uintptr_t)ptr, (idx < 1) + 1); }
```

Set the data pointer for a buffer descriptor.

#### Parameters

<i>idx</i>	Index of the buffer descriptor.
<i>ptr</i>	Pointer to the data buffer.

### eth\_set\_ready

```
#define eth_set_ready(  
    idx,  
    enable)
```

#### Value:

```
{  
    iob_eth_csrs_set_bd(iob_eth_csrs_get_bd(idx < 1) & ~TX_BD_READY |  
        (enable ? TX_BD_READY : 0),  
        idx < 1);  
}
```

Set the ready flag for a transmit buffer descriptor.

#### Parameters

<i>idx</i>	Index of the buffer descriptor.
<i>enable</i>	1 to set as ready, 0 otherwise.

## eth\_set\_wr

```
#define eth_set_wr(  
    idx,  
    enable)
```

### Value:

```
{  
    iob_eth_csrs_set_bd(iob_eth_csrs_get_bd(idx < 1) & ~TX_BD_WRAP |  
        (enable ? TX_BD_WRAP : 0),  
        idx < 1);  
}
```

Set the wrap flag for a buffer descriptor.

### Parameters

<i>idx</i>	Index of the buffer descriptor.
<i>enable</i>	1 to set the wrap flag, 0 otherwise.

## eth\_tx\_ready

```
#define eth_tx_ready(  
    idx)
```

### Value:

```
!((iob_eth_csrs_get_bd(idx < 1) & TX_BD_READY) || 0)
```

Check if the transmit buffer descriptor is ready.

### Parameters

<i>idx</i>	Index of the buffer descriptor.
------------	---------------------------------

### Returns

1 if ready, 0 otherwise.

## 4.6 iob\_eth\_macros.h File Reference

Macros for IOb-Eth core.

```
#include "iob_eth_conf.h"  
#include "iob_eth_rmac.h"
```

## Macros

### Mode Register Flags

- #define MODER\_RXEN (1 << 0)
- #define MODER\_TXEN (1 << 1)
- #define MODER\_NOPRE (1 << 2)
- #define MODER\_BRO (1 << 3)
- #define MODER\_IAM (1 << 4)
- #define MODER\_PRO (1 << 5)
- #define MODER\_IFG (1 << 6)
- #define MODER\_LOOP (1 << 7)
- #define MODER\_NBO (1 << 8)
- #define MODER\_EDE (1 << 9)
- #define MODER\_FULLLD (1 << 10)
- #define MODER\_RESET (1 << 11)
- #define MODER\_DCRC (1 << 12)
- #define MODER\_CRC (1 << 13)
- #define MODER\_HUGE (1 << 14)
- #define MODER\_PAD (1 << 15)
- #define MODER\_RSM (1 << 16)

### Interrupt Source and Mask Registers

- #define INT\_MASK\_TXF (1 << 0)
- #define INT\_MASK\_TXE (1 << 1)
- #define INT\_MASK\_RXF (1 << 2)
- #define INT\_MASK\_RXE (1 << 3)
- #define INT\_MASK\_BUSY (1 << 4)
- #define INT\_MASK\_TXC (1 << 5)
- #define INT\_MASK\_RXC (1 << 6)
- #define INT\_MASK\_TX (INT\_MASK\_TXF | INT\_MASK\_TXE)
- #define INT\_MASK\_RX (INT\_MASK\_RXF | INT\_MASK\_RXE)
- #define INT\_MASK\_ALL

*Mask for all interrupt sources.*

### Packet Length Register

- #define PACKETLEN\_MIN(min)
- #define PACKETLEN\_MAX(max)
- #define PACKETLEN\_MIN\_MAX(min, max)

### Transmit Buffer Number Register

- #define TX\_BD\_NUM\_VAL(x)

### Control Module Mode Register

- #define CTRLMODER\_PASSALL (1 << 0)
- #define CTRLMODER\_RXFLOW (1 << 1)
- #define CTRLMODER\_TXFLOW (1 << 2)

### MII Mode Register

- #define MIIMODER\_CLKDIV(x)

- #define MIIMODER\_NOPRE (1 << 8)

### **MII Command Register**

- #define MIICOMMAND\_SCAN (1 << 0)
- #define MIICOMMAND\_READ (1 << 1)
- #define MIICOMMAND\_WRITE (1 << 2)

### **MII Address Register**

- #define MIIADDRESS\_FIAD(x)
- #define MIIADDRESS\_RGAD(x)
- #define MIIADDRESS\_ADDR(phy, reg)

### **MII Transmit Data Register**

- #define MIITX\_DATA\_VAL(x)

### **MII Receive Data Register**

- #define MIIRX\_DATA\_VAL(x)

### **MII Status Register**

- #define MIISTATUS\_LINKFAIL (1 << 0)
- #define MIISTATUS\_BUSY (1 << 1)
- #define MIISTATUS\_INVALID (1 << 2)

### **TX Buffer Descriptor**

- #define TX\_BD\_CS (1 << 0)
- #define TX\_BD\_DF (1 << 1)
- #define TX\_BD\_LC (1 << 2)
- #define TX\_BD\_RL (1 << 3)
- #define TX\_BD\_RETRY\_MASK (0x00f0)
- #define TX\_BD\_RETRY(x)
- #define TX\_BD\_UR (1 << 8)
- #define TX\_BD\_CRC (1 << 11)
- #define TX\_BD\_PAD (1 << 12)
- #define TX\_BD\_WRAP (1 << 13)
- #define TX\_BD\_IRQ (1 << 14)
- #define TX\_BD\_READY (1 << 15)
- #define TX\_BD\_LEN(x)
- #define TX\_BD\_LEN\_MASK (0xffff << 16)
- #define TX\_BD\_STATS

### **RX Buffer Descriptor**

- #define RX\_BD\_LC (1 << 0)
- #define RX\_BD\_CRC (1 << 1)
- #define RX\_BD\_SF (1 << 2)
- #define RX\_BD\_TL (1 << 3)
- #define RX\_BD\_DN (1 << 4)

- #define RX\_BD\_IS (1 << 5)
- #define RX\_BD\_OR (1 << 6)
- #define RX\_BD\_MISS (1 << 7)
- #define RX\_BD\_CF (1 << 8)
- #define RX\_BD\_WRAP (1 << 13)
- #define RX\_BD\_IRQ (1 << 14)
- #define RX\_BD\_EMPTY (1 << 15)
- #define RX\_BD\_LEN(x)
- #define RX\_BD\_STATS

### MAC Address

- #define ETH\_MAC\_ADDR 0x01606e11020f

### Rx Return Codes

- #define ETH\_INVALID\_CRC -2
- #define ETH\_NO\_DATA -1
- #define ETH\_DATA\_RCV 0

### Frame Template Pointers

- #define MAC\_DEST\_PTR 0
- #define MAC\_SRC\_PTR (MAC\_DEST\_PTR + IOB\_ETH\_MAC\_ADDR\_LEN)
- #define ETH\_TYPE\_PTR (MAC\_SRC\_PTR + IOB\_ETH\_MAC\_ADDR\_LEN)
- #define PAYLOAD\_PTR (ETH\_TYPE\_PTR + 2)
- #define TEMPLATE\_LEN (PAYLOAD\_PTR)
- #define DWORD\_ALIGN(val)

### Debug

- #define ETH\_DEBUG\_PRINT 1

### RMAC Address

- #define ETH\_RMAC\_ADDR ETH\_MAC\_ADDR

## 4.6.1 Detailed Description

Macros for IOb-Eth core.

This file contains macros for various registers and flags in the IOb-Eth core.

## 4.6.2 Macro Definition Documentation

### CTRLMODER\_PASSALL

```
#define CTRLMODER_PASSALL (1 << 0)
```

Pass all receive frames



## **CTRLMODER\_RXFLOW**

```
#define CTRLMODER_RXFLOW (1 << 1)
```

Receive control flow

## **CTRLMODER\_TXFLOW**

```
#define CTRLMODER_TXFLOW (1 << 2)
```

Transmit control flow

## **DWORD\_ALIGN**

```
#define DWORD_ALIGN(  
    val)
```

### **Value:**

```
((val + 0x3) & ~0x3)
```

Dword alignment

## **ETH\_DATA\_RCV**

```
#define ETH_DATA_RCV 0
```

Data received

## **ETH\_DEBUG\_PRINT**

```
#define ETH_DEBUG_PRINT 1
```

Enable debug prints

## **ETH\_INVALID\_CRC**

```
#define ETH_INVALID_CRC -2
```

Invalid CRC

## **ETH\_MAC\_ADDR**

```
#define ETH_MAC_ADDR 0x01606e11020f
```

Default MAC address

## **ETH\_NO\_DATA**

```
#define ETH_NO_DATA -1
```

No data

## **ETH\_RMAC\_ADDR**

```
#define ETH_RMAC_ADDR ETH_MAC_ADDR
```

Default RMAC address if not defined

## **ETH\_TYPE\_PTR**

```
#define ETH_TYPE_PTR (MAC_SRC_PTR + IOB_ETH_MAC_ADDR_LEN)
```

Ethernet type pointer

## **INT\_MASK\_ALL**

```
#define INT_MASK_ALL
```

### **Value:**

```
(INT_MASK_TXF | INT_MASK_TXE | INT_MASK_RXF | INT_MASK_RXE | INT_MASK_TXC | \  
INT_MASK_RXC | INT_MASK_BUSY)
```

Mask for all interrupt sources.

## **INT\_MASK\_BUSY**

```
#define INT_MASK_BUSY (1 << 4)
```

Busy

## **INT\_MASK\_RX**

```
#define INT_MASK_RX (INT_MASK_RXF | INT_MASK_RXE)
```

Receive interrupts

## **INT\_MASK\_RXC**

```
#define INT_MASK_RXC (1 << 6)
```

Receive control frame

## **INT\_MASK\_RXE**

```
#define INT_MASK_RXE (1 << 3)
```

Receive error

## **INT\_MASK\_RXF**

```
#define INT_MASK_RXF (1 << 2)
```

Receive frame

## **INT\_MASK\_TX**

```
#define INT_MASK_TX (INT_MASK_TXF | INT_MASK_TXE)
```

Transmit interrupts

## **INT\_MASK\_TXC**

```
#define INT_MASK_TXC (1 << 5)
```

Transmit control frame

## **INT\_MASK\_TXE**

```
#define INT_MASK_TXE (1 << 1)
```

Transmit error

## **INT\_MASK\_TXF**

```
#define INT_MASK_TXF (1 << 0)
```

Transmit frame

## **MAC\_DEST\_PTR**

```
#define MAC_DEST_PTR 0
```

Destination MAC address pointer

## **MAC\_SRC\_PTR**

```
#define MAC_SRC_PTR (MAC_DEST_PTR + IOB_ETH_MAC_ADDR_LEN)
```

Source MAC address pointer

## **MIIADDRESS\_ADDR**

```
#define MIIADDRESS_ADDR(  
    phy,  
    reg)
```

### **Value:**

$(\text{MIIADDRESS\_FIAD}(\text{phy}) \mid \text{MIIADDRESS\_RGAD}(\text{reg}))$

PHY and register address \

## **MIIADDRESS\_FIAD**

```
#define MIIADDRESS_FIAD(  
    x)
```

### **Value:**

$((\langle x \rangle \& 0x1f) \ll 0)$

PHY address

## **MIIADDRESS\_RGAD**

```
#define MIIADDRESS_RGAD(  
    x)
```

**Value:**

`((x)&0x1f) < 8)`

Register address

**MIICOMMAND\_READ**

`#define MIICOMMAND_READ (1 << 1)`

Read status

**MIICOMMAND\_SCAN**

`#define MIICOMMAND_SCAN (1 << 0)`

Scan status

**MIICOMMAND\_WRITE**

`#define MIICOMMAND_WRITE (1 << 2)`

Write control data

**MIIMODER\_CLKDIV**

`#define MIIMODER_CLKDIV(  
    x)`

**Value:**

`((x)&0xfe)`

Clock divider - needs to be an even number

**MIIMODER\_NOPRE**

`#define MIIMODER_NOPRE (1 << 8)`

No preamble

**MIIRX\_DATA\_VAL**

`#define MIIRX_DATA_VAL(  
    x)`

**Value:**

((x)&0xffff)

Receive data

**MIISTATUS\_BUSY**

```
#define MIISTATUS_BUSY (1 << 1)
```

Busy

**MIISTATUS\_INVALID**

```
#define MIISTATUS_INVALID (1 << 2)
```

Invalid data

**MIISTATUS\_LINKFAIL**

```
#define MIISTATUS_LINKFAIL (1 << 0)
```

Link fail

**MIITX\_DATA\_VAL**

```
#define MIITX_DATA_VAL(  
    x)
```

**Value:**

((x)&0xffff)

Transmit data

**MODER\_BRO**

```
#define MODER_BRO (1 << 3)
```

Broadcast address

**MODER\_CRC**

```
#define MODER_CRC (1 << 13)
```

CRC enable

## **MODER\_DCRC**

```
#define MODER_DCRC (1 << 12)
```

Delayed CRC enable

## **MODER\_EDE**

```
#define MODER_EDE (1 << 9)
```

Excess defer enable

## **MODER\_FULLD**

```
#define MODER_FULLD (1 << 10)
```

Full duplex

## **MODER\_HUGE**

```
#define MODER_HUGE (1 << 14)
```

Huge packets enable

## **MODER\_IAM**

```
#define MODER_IAM (1 << 4)
```

Individual address mode

## **MODER\_IFG**

```
#define MODER_IFG (1 << 6)
```

Interframe gap for incoming frames

## **MODER\_LOOP**

```
#define MODER_LOOP (1 << 7)
```

Loopback

## **MODER\_NBO**

```
#define MODER_NBO (1 << 8)
```

No back-off

## **MODER\_NOPRE**

```
#define MODER_NOPRE (1 << 2)
```

No preamble

## **MODER\_PAD**

```
#define MODER_PAD (1 << 15)
```

Padding enabled

## **MODER\_PRO**

```
#define MODER_PRO (1 << 5)
```

Promiscuous mode

## **MODER\_RESET**

```
#define MODER_RESET (1 << 11)
```

FIXME: reset (undocumented)

## **MODER\_RSM**

```
#define MODER_RSM (1 << 16)
```

Receive small packets

## **MODER\_RXEN**

```
#define MODER_RXEN (1 << 0)
```



Receive enable

## **MODER\_TXEN**

```
#define MODER_TXEN (1 << 1)
```

Transmit enable

## **PACKETLEN\_MAX**

```
#define PACKETLEN_MAX(  
    max)
```

### **Value:**

```
((max)&0xffff) < 0)
```

Maximum packet length

## **PACKETLEN\_MIN**

```
#define PACKETLEN_MIN(  
    min)
```

### **Value:**

```
((min)&0xffff) < 16)
```

Minimum packet length \

## **PACKETLEN\_MIN\_MAX**

```
#define PACKETLEN_MIN_MAX(  
    min,  
    max)
```

### **Value:**

```
(PACKETLEN_MIN(min) | PACKETLEN_MAX(max))
```

Min and max packet length

## **PAYLOAD\_PTR**

```
#define PAYLOAD_PTR (ETH_TYPE_PTR + 2)
```

Payload pointer

## **RX\_BD\_CF**

```
#define RX_BD_CF (1 << 8)
```

Control frame

## **RX\_BD\_CRC**

```
#define RX_BD_CRC (1 << 1)
```

RX CRC error

## **RX\_BD\_DN**

```
#define RX_BD_DN (1 << 4)
```

Dribble nibble

## **RX\_BD\_EMPTY**

```
#define RX_BD_EMPTY (1 << 15)
```

RX buffer empty

## **RX\_BD\_IRQ**

```
#define RX_BD_IRQ (1 << 14)
```

Interrupt request enable

## **RX\_BD\_IS**

```
#define RX_BD_IS (1 << 5)
```

Invalid symbol

## **RX\_BD\_LC**

```
#define RX_BD_LC (1 << 0)
```

Late collision

## **RX\_BD\_LEN**

```
#define RX_BD_LEN(  
    x)
```

### **Value:**

```
((x)&0xffff) < 16)
```

Length of the buffer

## **RX\_BD\_MISS**

```
#define RX_BD_MISS (1 << 7)
```

Miss

## **RX\_BD\_OR**

```
#define RX_BD_OR (1 << 6)
```

Receiver overrun

## **RX\_BD\_SF**

```
#define RX_BD_SF (1 << 2)
```

Short frame

## **RX\_BD\_STATS**

```
#define RX_BD_STATS
```

### **Value:**

```
(RX_BD_LC | RX_BD_CRC | RX_BD_SF | RX_BD_TL | RX_BD_DN | RX_BD_IS |  
RX_BD_OR | RX_BD_MISS)
```

RX buffer descriptor statistics

## **RX\_BD\_TL**

```
#define RX_BD_TL (1 << 3)
```

Too long

## **RX\_BD\_WRAP**

```
#define RX_BD_WRAP (1 << 13)
```

Wrap

## **TEMPLATE\_LEN**

```
#define TEMPLATE_LEN (PAYLOAD_PTR)
```

Template length

## **TX\_BD\_CRC**

```
#define TX_BD_CRC (1 << 11)
```

TX CRC enable

## **TX\_BD\_CS**

```
#define TX_BD_CS (1 << 0)
```

Carrier sense lost

## **TX\_BD\_DF**

```
#define TX_BD_DF (1 << 1)
```

Defer indication

## **TX\_BD\_IRQ**

```
#define TX_BD_IRQ (1 << 14)
```

Interrupt request enable

## **TX\_BD\_LC**

```
#define TX_BD_LC (1 << 2)
```

Late collision

## **TX\_BD\_LEN**

```
#define TX_BD_LEN(  
    x)
```

### **Value:**

```
((x)&0xffff) < 16)
```

Length of the buffer

## **TX\_BD\_LEN\_MASK**

```
#define TX_BD_LEN_MASK (0xffff << 16)
```

Mask for the length of the buffer \

## **TX\_BD\_NUM\_VAL**

```
#define TX_BD_NUM_VAL(  
    x)
```

### **Value:**

```
((x) <= 0x80) ? (x) : 0x80)
```

Transmit buffer descriptor number

## **TX\_BD\_PAD**

```
#define TX_BD_PAD (1 << 12)
```

Pad enable for short packets

## **TX\_BD\_READY**

```
#define TX_BD_READY (1 << 15)
```

TX buffer ready

## **TX\_BD\_RETRY**

```
#define TX_BD_RETRY(  
    x)
```

### **Value:**

```
((x)&0x00f0) > 4)
```

Retry count

## **TX\_BD\_RETRY\_MASK**

```
#define TX_BD_RETRY_MASK (0x00f0)
```

Retry mask

## **TX\_BD\_RL**

```
#define TX_BD_RL (1 << 3)
```

Retransmission limit

## **TX\_BD\_STATS**

```
#define TX_BD_STATS
```

### **Value:**

```
(TX_BD_CS | TX_BD_DF | TX_BD_LC | TX_BD_RL | TX_BD_RETRY_MASK |  
TX_BD_UR) \
```

TX buffer descriptor statistics

## **TX\_BD\_UR**

```
#define TX_BD_UR (1 << 8)
```

Transmitter underrun

## **TX\_BD\_WRAP**

```
#define TX_BD_WRAP (1 << 13)
```

Wrap

## **4.7 iob\_eth\_rmac.h File Reference**

Remote MAC address (RMAC) definition.

## Detailed Description

Remote MAC address (RMAC) definition.

This file can be overwritten to define the `IOB_ETH_RMAC` macro. If not defined, it defaults to the local MAC address, effectively putting the core in a loopback mode (`IOB_ETH_RMAC = IOB_ETH_MAC`).

## 5 Linux Software

This section describes the Linux driver for the `iob_eth` peripheral. It includes details about the main kernel module, the kernel-user space interfaces it provides for interacting with the peripheral's Control and Status Registers (CSRs), and the available tests to verify the driver's functionality.

The driver consists of:

- A kernel module, implemented in `iob_eth_main.c`, which is the core of the driver.
- Three distinct kernel-user space interfaces: `/dev`, `ioctl`, and `sysfs`.
- A set of user space functions with a common API to access the CSRs through any of the interfaces.
- A test suite to validate the driver and the interfaces.

### 5.1 Kernel Module

The main source code for the kernel module is located in the `iob_eth_main.c` file. This module is implemented as a platform driver, which is responsible for probing and initializing the peripheral device based on information from the device tree. When the device is detected, the driver maps the peripheral's memory-mapped registers and creates the necessary user space interfaces (`/dev`, `ioctl`, and `sysfs`). It also implements the file operations (e.g., `read`, `write`, `ioctl`) for the `/dev` and `ioctl` interfaces.

### 5.2 User Space Interfaces

The driver provides three distinct interfaces for user space applications to interact with the `iob_eth` peripheral: `/dev`, `ioctl`, and `sysfs`. All three interfaces use a common set of user space functions to access the CSRs, with function prototypes that are similar to those of the bare-metal drivers, providing a consistent API. The baremetal function prototypes are documented in Section 4.

The following header files must be included in your user space application to use the API:

- `iob_eth_driver_files.h`
- `iob_eth_csrs.h`

Before using any of the API functions, you must initialize the library by calling the following function:

```
void iob_eth_csrs_init_baseaddr(uint32_t addr);
```

For the `/dev` and `ioctl` interfaces, this function opens the device file. For the `sysfs` interface, this function does nothing.

The following sections describe each of these interfaces in detail.

### 5.2.1 `/dev` Interface

The `/dev` interface allows direct access to the peripheral's registers through the device file `/dev/iob_eth`. Access to the CSRs is performed by seeking to the appropriate address offset using `lseek()` and then using `read()` or `write()` to access the register.

The following CSRs are available through this interface:

- `moder`: Address: `IOB_ETH_CSRS_MODER_ADDR`, Access: RW
- `int_source`: Address: `IOB_ETH_CSRS_INT_SOURCE_ADDR`, Access: RW
- `int_mask`: Address: `IOB_ETH_CSRS_INT_MASK_ADDR`, Access: RW
- `ipgt`: Address: `IOB_ETH_CSRS_IPGT_ADDR`, Access: RW
- `ipgr1`: Address: `IOB_ETH_CSRS_IPGR1_ADDR`, Access: RW
- `ipgr2`: Address: `IOB_ETH_CSRS_IPGR2_ADDR`, Access: RW
- `packetlen`: Address: `IOB_ETH_CSRS_PACKETLEN_ADDR`, Access: RW
- `collconf`: Address: `IOB_ETH_CSRS_COLLCONF_ADDR`, Access: RW
- `tx_bd_num`: Address: `IOB_ETH_CSRS_TX_BD_NUM_ADDR`, Access: RW
- `ctrlmoder`: Address: `IOB_ETH_CSRS_CTRLMODER_ADDR`, Access: RW
- `miimoder`: Address: `IOB_ETH_CSRS_MIIMODER_ADDR`, Access: RW
- `miicommand`: Address: `IOB_ETH_CSRS_MIICOMMAND_ADDR`, Access: RW
- `miiaddress`: Address: `IOB_ETH_CSRS_MIIADDRESS_ADDR`, Access: RW
- `miitx_data`: Address: `IOB_ETH_CSRS_MIITX_DATA_ADDR`, Access: RW
- `miirx_data`: Address: `IOB_ETH_CSRS_MIIRX_DATA_ADDR`, Access: RW
- `miistatus`: Address: `IOB_ETH_CSRS_MIISTATUS_ADDR`, Access: RW
- `mac_addr0`: Address: `IOB_ETH_CSRS_MAC_ADDR0_ADDR`, Access: RW
- `mac_addr1`: Address: `IOB_ETH_CSRS_MAC_ADDR1_ADDR`, Access: RW
- `eth_hash0_adr`: Address: `IOB_ETH_CSRS_ETH_HASH0_ADR_ADDR`, Access: RW
- `eth_hash1_adr`: Address: `IOB_ETH_CSRS_ETH_HASH1_ADR_ADDR`, Access: RW
- `eth_txctrl`: Address: `IOB_ETH_CSRS_ETH_TXCTRL_ADDR`, Access: RW
- `tx_bd_cnt`: Address: `IOB_ETH_CSRS_TX_BD_CNT_ADDR`, Access: R



- `rx_bd_cnt`: Address: `IOB_ETH_CSRS_RX_BD_CNT_ADDR`, Access: R
- `tx_word_cnt`: Address: `IOB_ETH_CSRS_TX_WORD_CNT_ADDR`, Access: R
- `rx_word_cnt`: Address: `IOB_ETH_CSRS_RX_WORD_CNT_ADDR`, Access: R
- `rx_nbytes`: Address: `IOB_ETH_CSRS_RX_NBYTES_ADDR`, Access: R
- `frame_word`: Address: `IOB_ETH_CSRS_FRAME_WORD_ADDR`, Access: RW
- `phy_rst_val`: Address: `IOB_ETH_CSRS_PHY_RST_VAL_ADDR`, Access: R
- `bd`: Address: `IOB_ETH_CSRS_BD_ADDR`, Access: RW
- `version`: Address: `IOB_ETH_CSRS_VERSION_ADDR`, Access: R

### 5.2.2 ioctl Interface

The `ioctl` interface uses I/O control commands to interact with the peripheral. The function prototypes provided for this interface are identical to the `/dev` interface functions.

The following IOCTL commands are defined for each CSR:

- `WR_MODER`: Write to the moder CSR.
- `RD_MODER`: Read from the moder CSR.
- `WR_INT_SOURCE`: Write to the int\_source CSR.
- `RD_INT_SOURCE`: Read from the int\_source CSR.
- `WR_INT_MASK`: Write to the int\_mask CSR.
- `RD_INT_MASK`: Read from the int\_mask CSR.
- `WR_IPGT`: Write to the ipgt CSR.
- `RD_IPGT`: Read from the ipgt CSR.
- `WR_IPGR1`: Write to the ipgr1 CSR.
- `RD_IPGR1`: Read from the ipgr1 CSR.
- `WR_IPGR2`: Write to the ipgr2 CSR.
- `RD_IPGR2`: Read from the ipgr2 CSR.
- `WR_PACKETLEN`: Write to the packetlen CSR.
- `RD_PACKETLEN`: Read from the packetlen CSR.
- `WR_COLLCONF`: Write to the collconf CSR.
- `RD_COLLCONF`: Read from the collconf CSR.
- `WR_TX_BD_NUM`: Write to the tx\_bd\_num CSR.
- `RD_TX_BD_NUM`: Read from the tx\_bd\_num CSR.
- `WR_CTRLMODER`: Write to the ctrlmoder CSR.

- RD\_CTRLMODER: Read from the ctrlmoder CSR.
- WR\_MIIMODER: Write to the miimoder CSR.
- RD\_MIIMODER: Read from the miimoder CSR.
- WR\_MIICOMMAND: Write to the miicommand CSR.
- RD\_MIICOMMAND: Read from the miicommand CSR.
- WR\_MIIADDRESS: Write to the miiaddress CSR.
- RD\_MIIADDRESS: Read from the miiaddress CSR.
- WR\_MIITX\_DATA: Write to the miitx\_data CSR.
- RD\_MIITX\_DATA: Read from the miitx\_data CSR.
- WR\_MIIRX\_DATA: Write to the miirx\_data CSR.
- RD\_MIIRX\_DATA: Read from the miirx\_data CSR.
- WR\_MIISTATUS: Write to the miistatus CSR.
- RD\_MIISTATUS: Read from the miistatus CSR.
- WR\_MAC\_ADDR0: Write to the mac\_addr0 CSR.
- RD\_MAC\_ADDR0: Read from the mac\_addr0 CSR.
- WR\_MAC\_ADDR1: Write to the mac\_addr1 CSR.
- RD\_MAC\_ADDR1: Read from the mac\_addr1 CSR.
- WR\_ETH\_HASH0\_ADR: Write to the eth\_hash0\_adr CSR.
- RD\_ETH\_HASH0\_ADR: Read from the eth\_hash0\_adr CSR.
- WR\_ETH\_HASH1\_ADR: Write to the eth\_hash1\_adr CSR.
- RD\_ETH\_HASH1\_ADR: Read from the eth\_hash1\_adr CSR.
- WR\_ETH\_TXCTRL: Write to the eth\_txctrl CSR.
- RD\_ETH\_TXCTRL: Read from the eth\_txctrl CSR.
- RD\_TX\_BD\_CNT: Read from the tx\_bd\_cnt CSR.
- RD\_RX\_BD\_CNT: Read from the rx\_bd\_cnt CSR.
- RD\_TX\_WORD\_CNT: Read from the tx\_word\_cnt CSR.
- RD\_RX\_WORD\_CNT: Read from the rx\_word\_cnt CSR.
- RD\_RX\_NBYTES: Read from the rx\_nbytes CSR.
- WR\_FRAME\_WORD: Write to the frame\_word CSR.
- RD\_FRAME\_WORD: Read from the frame\_word CSR.
- RD\_PHY\_RST\_VAL: Read from the phy\_rst\_val CSR.
- WR\_BD: Write to the bd CSR.
- RD\_BD: Read from the bd CSR.
- RD\_VERSION: Read from the version CSR.

### 5.2.3 sysfs Interface

The `sysfs` interface exposes the peripheral's registers as files in the system's file system. The functions prototypes provided for this interface are identical to the `/dev` interface functions.

The CSRs are exposed as files in the following directory:

```
/sys/class/iob_eth/iob_eth/
```

The following files are available for each CSR:

- `moder`: Access the moder CSR. (Mode: RW)
- `int_source`: Access the int\_source CSR. (Mode: RW)
- `int_mask`: Access the int\_mask CSR. (Mode: RW)
- `ipgt`: Access the ipgt CSR. (Mode: RW)
- `ipgr1`: Access the ipgr1 CSR. (Mode: RW)
- `ipgr2`: Access the ipgr2 CSR. (Mode: RW)
- `packetlen`: Access the packetlen CSR. (Mode: RW)
- `collconf`: Access the collconf CSR. (Mode: RW)
- `tx_bd_num`: Access the tx\_bd\_num CSR. (Mode: RW)
- `ctrlmoder`: Access the ctrlmoder CSR. (Mode: RW)
- `miimoder`: Access the miimoder CSR. (Mode: RW)
- `miicommand`: Access the miicommand CSR. (Mode: RW)
- `miiaddress`: Access the miiaddress CSR. (Mode: RW)
- `miitx_data`: Access the miitx\_data CSR. (Mode: RW)
- `miirx_data`: Access the miirx\_data CSR. (Mode: RW)
- `miistatus`: Access the miistatus CSR. (Mode: RW)
- `mac_addr0`: Access the mac\_addr0 CSR. (Mode: RW)
- `mac_addr1`: Access the mac\_addr1 CSR. (Mode: RW)
- `eth_hash0_adr`: Access the eth\_hash0\_adr CSR. (Mode: RW)
- `eth_hash1_adr`: Access the eth\_hash1\_adr CSR. (Mode: RW)
- `eth_txctrl`: Access the eth\_txctrl CSR. (Mode: RW)
- `tx_bd_cnt`: Access the tx\_bd\_cnt CSR. (Mode: R)
- `rx_bd_cnt`: Access the rx\_bd\_cnt CSR. (Mode: R)
- `tx_word_cnt`: Access the tx\_word\_cnt CSR. (Mode: R)
- `rx_word_cnt`: Access the rx\_word\_cnt CSR. (Mode: R)

- `rx_nbytes`: Access the `rx_nbytes` CSR. (Mode: R)
- `frame_word`: Access the `frame_word` CSR. (Mode: RW)
- `phy_rst_val`: Access the `phy_rst_val` CSR. (Mode: R)
- `bd`: Access the `bd` CSR. (Mode: RW)
- `version`: Access the `version` CSR. (Mode: R)

## 5.3 User Space Application

User space applications can be developed to interact with the peripheral's driver interfaces. An example application, `user/iob_eth_user.c`, is provided for some cores. Otherwise, the auto-generated test application, `user/iob_eth_tests.c`, can serve as a reference for creating custom user space applications.

**Building an application** User space applications can be built using the `Makefile` located in the `user` directory. You need to specify the name of your application's source file (without the `.c` extension) and the desired interface.

```
make BIN=<your_app_name> IF=<interface>
```

The `IF` variable can be set to `sysfs`, `dev`, or `ioctl` to build the application for the corresponding interface. For example, to build an application from a source file named `my_app.c`, you would run `make BIN=my_app IF=sysfs`.

**Running the application** To run the application, execute the compiled binary in the target Linux system, replacing `<your_app_name>` and `<interface>` with the ones you selected during the build:

```
./<your_app_name>_<interface>
```

## 5.4 Tests

A test suite is provided to verify the functionality and performance of the driver interfaces. The test source code is located in `user/iob_eth_tests.c`.

**Building the tests** The tests can be built using the `Makefile` in the `user` directory by setting the `BIN` variable to `iob_eth_tests`:

```
make BIN=iob_eth_tests IF=<interface>
```

The `IF` variable can be set to `sysfs`, `dev`, or `ioctl` to test the corresponding interface.

**Running the tests** To run the tests, execute the compiled binary in the target Linux system, replacing <interface> with the one you selected during build:

```
./iob_eth_tests_<interface>
```

The test suite includes:

- **Functionality tests:** Verify that writing to and reading from Control and Status Registers (CSRs) works correctly.
- **Error Handling tests:** Simulate faults and verify that appropriate error messages are generated.
- **Performance tests:** Measure the time taken for a large number of read and write operations to evaluate the interface performance.

## 6 CSR Detailed Description for Software

This section presents a detailed description of the Control and Status Registers, described in section 2.4, useful for writing the software drivers of this IP core.

Bit #	R/W	Description
31-17		Reserved
16	RW	RECSMALL - Receive Small Packets. 0 = Packets smaller than MINFL are ignored; 1 = Packets smaller than MINFL are accepted.
15	RW	PAD - Padding enabled. 0 = do not add pads to short frames; 1 = add pads to short frames (until the minimum frame length is equal to MINFL).
14	RW	HUGEN = Huge Packets Enable. 0 = the maximum frame length is MAXFL. All Additional bytes are discarded; 1 = Frames up 64KB are transmitted.
13	RW	CRCEN - CRC Enable. 0 = Tx MAC does not append the CRC (passed frames already contain the CRC); 1 = Tx MAC appends the CRC to every frame.
12	RW	DLYCRCEN - Delayed CRC Enabled. 0 = Normal operation (CRC calculation starts immediately after the SFD); 1 = CRC calculation starts 4 bytes after the SFD.
11	RW	Reserved
10	RW	FULLD - Full Duplex. 0 = Half duplex mode; 1 = Full duplex mode.
9	RW	EXDFREN - Excess Defer Enabled. 0 = When the excessive deferral limit is reached, a packet is aborted; 1 = MAC waits for the carrier indefinitely.
8	RW	NOBCKOF - No Backoff. 0 = Normal operation (a binary exponential backoff algorithm is used); 1 = Tx MAC starts retransmitting immediately after the collision.
7	RW	LOOPBCK - Loop Back. 0 = Normal operation; 1 = Tx is looped back to the RX.

6	RW	IFG - Interframe Gap for Incoming frames. 0 = Normal operation (minimum IFG is required for a frame to be accepted); 1 = All frames are accepted regardless to the IFG.
5	RW	PRO - Promiscuous. 0 = Check the destination address of the incoming frames; 1 = Receive the frame regardless of its address.
4	RW	IAM - Individual Address Mode. 0 = Normal operation (physical address is checked when the frame is received); 1 = The individual hash table is used to check all individual addresses received.
3	RW	BRO - Broadcast Address. 0 = Receive all frames containing the broadcast address; 1 = Reject all frames containing the broadcast address unless the PRO bit=1.
2	RW	NOPRE - No Preamble. 0 = Normal operation (7-byte preamble); 1 = No preamble is sent.
1	RW	TXEN - Transmit Enable. 0 = Transmit is disabled; 1 = Transmit is enabled. If the value, written to the TX_BD_NUM register, is equal to 0x0 (zero buffer descriptors are used), then the transmitter is automatically disabled regardless of the TXEN bit.
0	RW	RXEN - Receive Enable. 0 = Transmit is disabled; 1 = Transmit is enabled. If the value, written to the TX_BD_NUM register, is equal to 0x80 (all buffer descriptors are used for transmit buffer descriptors, so there is no receive BD), then the receiver is automatically disabled regardless of the RXEN bit.

Table 72: MODER (Mode Register)

Bit #	R/W	Description
31-7		Reserved
6	RW	RXC - Receive Control Frame This bit indicates that the control frame was received. It is cleared by writing 1 to it. Bit RXFLOW (CTRLMODER register) must be set to 1 in order to get the RXC bit set.
5	RW	TXC - Transmit Control Frame This bit indicates that a control frame was transmitted. It is cleared by writing 1 to it. Bit RXFLOW (CTRLMODER register) must be set to 1 in order to get the TXC bit set.
4	RW	BUSY - Busy This bit indicates that a buffer was received and discarded due to a lack of buffers. It is cleared by writing 1 to it. This bit appears regardless to the IRQ bits in the Receive or Transmit Buffer Descriptors.
3	RW	RXE - Receive Error This bit indicates that an error occurred while receiving data. It is cleared by writing 1 to it. This bit appears only when IRQ bit is set in the Receive Buffer Descriptor.
2	RW	RXB - Receive Frame This bit indicates that a frame was received. It is cleared by writing 1 to it. This bit appears only when IRQ bit is set in the Receive Buffer Descriptor. If a control frame is received, then RXC bit is set instead of the RXB bit. (CTRLMODER (Control Module Mode Register) description for more details.)
1	RW	TXE - Transmit Error This bit indicates that a buffer was not transmitted due to a transmit error. It is cleared by writing 1 to it. This bit appears only when IRQ bit is set in the Receive Buffer Descriptor. This bit appears only when IRQ bit is set in the Transmit Buffer Descriptor.

0	RW	<b>TXB - Transmit Buffer</b> This bit indicates that a buffer has been transmitted. It is cleared by writing 1 to it. This bit appears only when IRQ bit is set in the Transmit Buffer Descriptor.
---	----	---

Table 73: INT\_SOURCE (Interrupt Source Register)

Bit #	R/W	Description
31-7		Reserved
6	RW	<b>RXC_M - Receive Control Frame Mask</b> 0 = Event masked 1 = Event causes an interrupt
5	RW	<b>TXC - Transmit Control Frame Mask</b> 0 = Event masked 1 = Event causes an interrupt
4	RW	<b>BUSY - Busy Mask</b> 0 = Event masked 1 = Event causes an interrupt
3	RW	<b>RXE - Receive Error Mask</b> 0 = Event masked 1 = Event causes an interrupt
2	RW	<b>RXB - Receive Frame Mask</b> 0 = Event masked 1 = Event causes an interrupt
1	RW	<b>TXE - Transmit Error Mask</b> 0 = Event masked 1 = Event causes an interrupt
0	RW	<b>TXB - Transmit Buffer Mask</b> 0 = Event masked 1 = Event causes an interrupt

Table 74: INT\_MASK (Interrupt Mask Register)

Bit #	R/W	Description
31-7		Reserved
6-0	RW	<b>IPGT - Back to Back Inter Packet Gap</b> Full Duplex: The recommended value is 0x15, which equals 0.96 $\mu$ s IPG (100 Mbps) or 9.6 $\mu$ s (10 Mbps). The desired period in nibble times minus 6 should be written to the register. Half Duplex: The recommended value is 0x12, which equals 0.96 $\mu$ s IPG (100 Mbps) or 9.6 $\mu$ s (10 Mbps). The desired period in nibble times minus 3 should be written to the register.

Table 75: IPGT (Back to Back Inter Packet Gap Register)

Bit #	R/W	Description
31-7		Reserved
6-0	RW	<b>IPGR1 - Non Back to Back Inter Packet Gap 1</b> When a carrier sense appears within the IPGR1 window, Tx MAC defers and the IPGR counter is reset. When a carrier sense appears later than the IPGR1 window, the IPGR counter continues counting. The recommended and default value for this register is 0xC. It must be within the range [0,IPGR2].

Table 76: IPGR1 (Non Back to Back Inter Packet Gap Register 1)

Bit #	R/W	Description
31-7		Reserved
6-0	RW	IPGR2 - Non Back to Back Inter Packet Gap 2 The recommended and default value is 0x12, which equals to $0.96\mu s$ IPG (100 Mbps) or $9.6\mu s$ (10 Mbps).

Table 77: IPGR2 (Non Back to Back Inter Packet Gap Register 2)

Bit #	R/W	Description
31-16	RW	MINFL - Minimum Frame Length The minimum Ethernet packet is 64 bytes long. If a reception of smaller frames is needed, assert the RECSMALL bit (in the mode register MODER) or change the value of this register. To transmit small packets, assert the PAD bit or the MINFL value (see the PAD bit description in the MODER register).
15-0	RW	MAXFL - Maximum Frame Length The maximum Ethernet packet is 1518 bytes long. To support this and to leave some additional space for the tags, a default maximum packet length equals 1536 bytes (0x0600). If there is a need to support bigger packets, you can assert the HUGEN bit or increase the value of the MAXFL field (see the HUGEN bit description in the MODER).

Table 78: PACKETLEN (Packet Length Register)

Bit #	R/W	Description
31-20		Reserved
19-16	RW	MAXRET - Maximum Retry This field specifies the maximum number of consequential retransmission attempts after the collision is detected. When the maximum number has been reached, the Tx MAC reports an error and stops transmitting the current packet. According to the Ethernet standard, the MAXRET default value is set to 0xf (15).
15-6	RW	Reserved
5-0	RW	COLLVALID - Collision Valid This field specifies a collision time window. A collision that occurs later than the time window is reported as a "Late Collisions" and transmission of the current packet is aborted. The default value equals 0x3f (by default, a late collision is every collision that occurs 64 bytes (63 + 1) from the preamble)

Table 79: COLLCONF (Collision and Retry Configuration Register)

Bit #	R/W	Description
31-8	RW	Reserved
7-0	RW	Transmit Buffer Descriptor (Tx BD) Number Number of the Tx BD. Number of the Rx BD equals to the (0x80 – Tx BD number). Maximum number of the Tx BD is 0x80. Values greater than 0x80 cannot be written to this register (ignored).

Table 80: TX\_BD\_NUM (Transmit BD Number Register)



Bit #	R/W	Description
31-3	RW	Reserved
2	RW	TXFLOW - Transmit Flow Control 0 = PAUSE control frames are blocked. 1 = PAUSE control frames are allowed to be sent. This bit enables the TXC bit in the INT_SOURCE register.
1	RW	RXFLOW - Receive Flow Control 0 = Received PAUSE control frames are ignored. 1 = The transmit function (Tx MAC) is blocked when a PAUSE control frame is received. This bit enables the RXC bit in the INT_SOURCE register.
0	RW	PASSALL - Pass All Receive Frames 0 = Control frames are not passed to the host. RXFLOW must be set to 1 in order to use PAUSE control frames. 1 = All received frames are passed to the host.

Table 81: CTRLMODER (Control Module Mode Register)

PASSALL #	RXFLOW	Description
0	0	When a PAUSE control frame is received, nothing happens. The control frame is not stored to the memory.
0	1	When a PAUSE control frame is received, RXC interrupt is set and pause timer is updated. The control frame is not stored to the memory.
1	0	When a PAUSE control frame is received, it is stored to the memory as a normal data frame. RXB interrupt is set (if the related buffer descriptor has an IRQ bit set to 1). RXC interrupt is not set and pause timer is not updated.
1	1	When a PAUSE control frame is received, RXC interrupt is set and pause timer is updated. Besides that the control frame is also stored to the memory as a normal data frame.

Table 82: PASSALL and RXFLOW Operation

Bit #	R/W	Description
31-9		Reserved
8	RW	MIINOPRE - No Preamble 0 = 32-bit preamble sent 1 = No preamble sent
7-0	RW	CLKDIV - Clock Divider The field is a host clock divider factor. The host clock can be divided by an even number, greater than 1. The default value is 0x64 (100).

Table 83: MIIMODER (MII Mode Register)

Bit #	R/W	Description
31-3		Reserved
2	RW	WCTRLDATA - Write Control Data
1	RW	RSTAT - Read Status
0	RW	SCANSTAT - Scan Status

Table 84: MIICOMMAND (MII Command Register)

**Note:** While one operation is in progress, BUSY signal (MIISTATUS register is set. Next operation can be started after the previous one is finished (and BUSY signal cleared to zero).

Bit #	R/W	Description
31-13		Reserved
12-8	RW	RGAD - Register Address (within the PHY selected by the FIAD[4:0])
7-5	RW	Reserved
4-0	RW	FIAD - PHY Address

Table 85: MIIADDRESS (MII Address Register)

Bit #	R/W	Description
31-16		Reserved
15-0	RW	CTRLDATA - Control Data (data to be written to the PHY)

Table 86: MIITX\_DATA (MII Transmit Data)

Bit #	R/W	Description
31-16		Reserved
15-0	R	PRSD - Received Data (data to be read from PHY)

Table 87: MIIRX\_DATA (MII Receive Data)

Bit #	R/W	Description
31-3		Reserved
2	R	NVALID - Invalid 0 = The data in the MSTATUS register is valid. 1 = The data in the MSTATUS register is invalid. This bit is only valid when the scan status operation is active.
1	R	BUSY 0 = The MII is ready. 1 = The MII is busy (operation in progress).
0	R	LINKFAIL 0 = The link is OK. 1 = The link failed. The link fail condition occurred (now the link might be OK). Another status read gets a new status.

Table 88: MIISTATUS (MII Status Register)

Bit #	R/W	Description
31-24	RW	Byte 2 of the Ethernet MAC address (individual address)
23-16	RW	Byte 3 of the Ethernet MAC address (individual address)
15-8	RW	Byte 4 of the Ethernet MAC address (individual address)
7-0	RW	Byte 5 of the Ethernet MAC address (individual address)

Table 89: MAC\_ADDR0 (MAC Address Register 0)

Bit #	R/W	Description
31-16	RW	Reserved
15-8	RW	Byte 0 of the Ethernet MAC address (individual address)
7-0	RW	Byte 1 of the Ethernet MAC address (individual address)

Table 90: MAC\_ADDR1 (MAC Address Register 1)

Note: When an address is transmitted, byte 0 is sent first and byte 5 last.

Bit #	R/W	Description
31-0	RW	Hash0 value

Table 91: HASH0 (HASH Register 0)

Bit #	R/W	Description
31-0	RW	Hash1 value

Table 92: HASH1 (HASH Register 1)

Bit #	R/W	Description
31-17		Reserved
16	RW	TXPAUSERQ - Tx Pause Request Writing 1 to this bit starts sending control frame procedure. Bit is automatically cleared to zero.
15-0	RW	TXPAUSETV - Tx Pause Timer Value The value that is send in the pause control frame.

Table 93: TXCTRL (Tx Control Register)

## 7 Buffer Descriptors

The buffer descriptors are 64 bits long. The first 32 bits are reserved for length and status while the last 32 bits contain the pointer to the associated buffer (where data is stored). The Ethernet MAC core has an internal RAM that can store up to 128 BDs (for both Rx and Tx).

The internal memory saves all descriptors at addresses from 0x400 to 0x7ff (128 64bit descriptors). The transmit descriptors are located between the start address (0x400) and the address that equals the value written in the TX\_BD\_NUM register (page 14) multiplied by 8. This register holds the number of the used Tx buffer descriptors. The receive descriptors are located between the start address (0x400), plus the address number written in the TX\_BD\_NUM multiplied by 8, and the descriptor end address (0x7ff). The transmit and receive status of the packet is written to the associated buffer descriptor once its transmission/reception is finished.

Bit #	R/W	Description
63-32	RW	TXPNT - Transmit Pointer This is the buffer pointer when the associated frame is stored.
31-16	RW	LEN - Length Number of bytes associated with the BD to be transmitted.
15	RW	RD - Tx BD Ready 0 = The buffer associated with this buffer descriptor is not ready, and you are free to manipulate it. After the data from the associated buffer has been transmitted or after an error condition occurred, this bit is cleared to 0. 1 = The data buffer is ready for transmission or is currently being transmitted. You are not allowed to manipulate this descriptor once this bit is set.
14	RW	IRQ - Interrupt Request Enable 0 = No interrupt is generated after the transmission. 1 = When data associated with this buffer descriptor is sent, a TXB or TXE interrupt will be asserted (INT_SOURCE (Interrupt Source Register) for more details).
13	RW	WR - Wrap 0 = This buffer descriptor is not the last descriptor in the buffer descriptor table. 1 = This buffer descriptor is the last descriptor in the buffer descriptor table. After this buffer descriptor was used, the first buffer descriptor in the table will be used again.
12	RW	PAD - Pad Enable 0 = No pads will be add at the end of short packets. 1 = Pads will be added to the end of short packets.
11	RW	CRC - CRC Enable 0 = CRC won't be added at the end of the packet. 1 = CRC will be added at the end of the packet.
10-9	RW	Reserved
8	RW	UR - Underrun Underrun occured while sending this buffer.
7-4	RW	RTRY - Retry Count This bit indicates the number of retries before the frame was successfully sent.
3	RW	RL - Retransmission Limit This bit is set when the transmitter fails. (Retry Limit + 1) attempts to successfully transmit a message due to repeated collisions on the medium. The Retry Limit is set in the COLLCONF register.
2	RW	LC - Late Collision Late collision occurred while sending this buffer. The transmission is stopped and this bit is written. Late collision is defined in the COLLCONF register.
1	RW	DF - Defer Indication The frame was deferred before being sent successfully, i.e. the transmitter had to wait for Carrier Sense before sending because the line was busy. This is not a collision indication. Collisions are indicated in RTRY.
0	RW	CS - Carrier Sense Lost This bit is set when Carrier Sense is lost during a frame transmission. The Ethernet controller writes CS after it finishes sending the buffer.

Table 94: Tx Buffer Descriptor

Bit #	R/W	Description
63-32	RW	RXPNT - Receive Pointer This is the pointer to the buffer storing the associated frame.
31-16	RW	LEN - Length Number of bytes associated with the BD to be transmitted.
15	RW	E - Empty 0 = The data buffer associated with this buffer descriptor has been filled with data or has stopped because an error occurred. The core can read or write this BD. As long as this bit is zero, this buffer descriptor won't be used. 1 = The data buffer is empty (and ready for receiving data) or currently receiving data.
14	RW	IRQ - Interrupt Request Enable 0 = No interrupt is generated after the reception. 1 = When data is received (or error occurs), an RXF interrupt will be asserted (See INT_SOURCE (Interrupt Source Register) for more details).
13	RW	WR - Wrap 0 = This buffer descriptor is not the last descriptor in the buffer descriptor table. 1 = This buffer descriptor is the last descriptor in the buffer descriptor table. After this buffer descriptor was used, the first buffer descriptor in the table will be used again.
12-9	RW	Reserved
8	RW	CF - Control Frame 0 = Normal data frame received. 1 = Control frame received.
7	RW	M - Miss 0 = The frame is received because of an address recognition hit. 1 = The frame is received because of promiscuous mode. The Ethernet controller sets M for frames that are accepted in promiscuous mode but are tagged as a miss by internal address recognition. Thus, in promiscuous mode, M determines whether a frame is destined for this station.
6	RW	OR - Overrun This bit is set when a receiver overrun occurs during frame reception.
5	RW	IS - Invalid Symbol This bit is set when the reception of an invalid symbol is detected by the PHY.
4	RW	DN - Dribble Nibble This bit is set when a received frame cannot be divided by 8 (one extra nibble has been received).
3	RW	TL - Too Long This bit is set when a received frame is too long (bigger than the value set in the PACKETLEN register).
2	RW	SF - Short Frame This bit is set when a frame that is smaller than the minimum length is received (minimum length is set in the PACKETLEN register).
1	RW	CRC - Rx CRC Error This bit is set when a received frame contains a CRC error.
0	RW	LC - Late Collision This bit is set when a late collision occurred while receiving a frame.

Table 95: Rx Buffer Descriptor