# High speed UDP transport protocol

# 1.Introduction

UDP is a connectionless transport layer protocol and belongs to the TCP/IP protocol cluster. UDP has the advantages of low resource consumption and high communication efficiency. It is usually used for real-time transmission of audio and video. This module based on Spinal HDL language, designed a maximum support bandwidth of 100G UDP communication circuit system.

# 2.Overview

User Datagram Protocol (UDP) is a connectionless transport layer Protocol. Connectionless Indicates that no logical connection is established between the sender and receiver during data transmission. The official Ethernet standard is IEEE 802.3, which defines the frame structure of data transmission over the Ethernet. Figure 2-1 shows the format of data transmission in a single packet. As shown in Figure 2-1, Ethernet packets encapsulate layer by layer protocols to transmit data. User data is packaged in UDP, which is based on IP, and IP is sent through the MAC layer. In terms of inclusion relation: The data segments in MAC frames are IP datagrams, the data segments in IP packets are UDP packets, and the data segments in UDP packets are the data that users want to transmit.

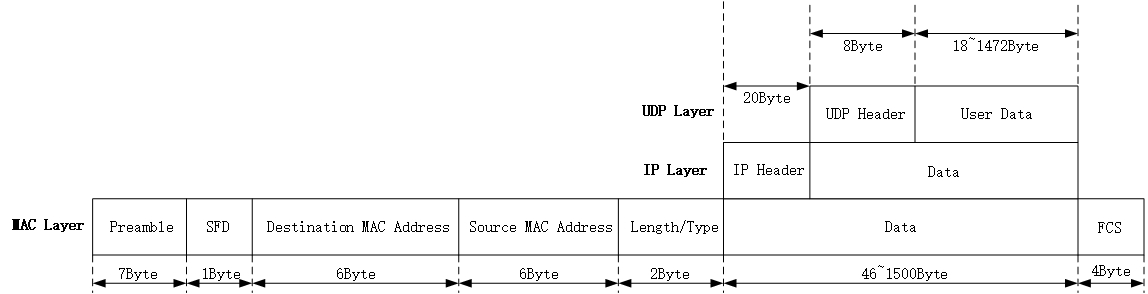


Figure 2-1: Ethernet UDP transmission packet format

### 2.1 MAC Layer:

Ethernet data is sent and received from beginning to end in the same order as above, and we explain the regions further below.

Preamble: In order to achieve the correct interpretation of the underlying data, the physical layer uses 7 bytes synchronization codes (alternating 0 and 1 (55-55-55-55-55)) to achieve data synchronization.

Frame Start Frame Delimiter (SFD): Use a one-byte SFD (fixed value 0xD5) to indicate the Start of a Frame, followed by the Ethernet Frame header.

Destination MAC address: indicates the physical MAC address of the receiving end, which occupies 6 bytes. MAC addresses are classified into unicast addresses, multicast addresses, and broadcast addresses. Unicast address: the lowest byte is 0, usually used to identify a unique device. Multicast address: the lowest byte of the first byte is 1. It is used to identify multiple devices belonging to the same group. Broadcast address: all 48 bits are 1, that is, FF-FF-FF-FF-FF, which is used to identify all devices in the same network segment.

Source MAC address: indicates the physical MAC address of the sender. It occupies 6 bytes.

Length/type: The length/type in the preceding figure has two meanings. If the value of the two bytes is less than 1536 (0x0600 in hexadecimal), it indicates the length of the data segment on the Ethernet. If the value of these two bytes is greater than 1536, it indicates which upper-layer protocol the data in the Ethernet belongs to.

Data: On the Ethernet, the minimum length of a data segment ranges from 46 to 1500 bytes. The Maximum value of 1500 is called the MTU (Maximum Transmission Unit) of the Ethernet.

Frame Check Sequence (FCS): To ensure the correct transmission of data, a 4-byte cyclic redundancy check (CRC Check) code is added to the end of the data to detect whether the data is transmitted incorrectly. CRC data check starts from the Ethernet frame header and does not contain the lead code and frame start delimiter.

### 2.2 IP Layer:

IP is the core protocol and carrier of TCP/IP. IP defines the basic unit and format of data transmission. As shown in Figure 2-1, the IP protocol is in the data segment of the Ethernet MAC frame format. The IP protocol content consists of the IP header and data fields. All TCP, UDP, and ICMP data is transmitted in the format of IP packets. Table 2-1 lists the format of IP packets.

Table 2-1:Format of IP frame headers

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | IHL | TOS | Total Length | |
| Identification | | | Flag | Fragment Offset |
| TTL | | Protocol | Header Checksum | |
| Source IP Address | | | | |
| Destination IP Address | | | | |
| Data | | | | |

Version: 4-bit IP Version. If the value is set to binary 0100, it indicates IPv4. If the value is set to binary 0110, it indicates IPv6.

Header Length: 4-bit Header Length (IHL, Internet Header Length), which indicates the number of 32-bit bits (4 bytes) in the IP Header. In the absence of optional fields, the LENGTH of the IP header is 20 bytes, so the value of the header length is 5.

Service Type: 8-bit Service Type (TOS), which is divided into two subfields: the 3-bit priority field (now largely ignored) and the 4-bit TOS field, with the last bit fixed at 0. If the service type is 0, it indicates common service.

Total Length: 16-bit Total Length of the IP packet, including the IP header and IP data part, in bytes. Since this field is 16 bits long, IP datagrams can be up to 65535 bytes long. Although in theory IP datagrams up to 65535 bytes can be transmitted, in practice there are factors such as the maximum carrying capacity of the network to be considered.

Identification field: The 16-bit Identification field identifies each datagram sent by the host. It is usually incremented by 1 each time a message is sent.

Flags field: 3-bit Flags field. The first bit is reserved. The second bit indicates that sharding is forbidden (1 indicates no sharding, 0 indicates sharding is allowed). Bit 3 identifies more shards (all but the last shard of the datagram are 1).

Fragment Offset: 13-bit Fragment Offset, used to identify the Fragment order when the receiver reassembles datagrams.

TTL: an 8-bit TTL field. The TTL (Time To Live) field prevents lost packets from being transmitted endlessly. It is usually set to 64 or 128.

Protocol: 8-bit Protocol: indicates the Protocol type used by the upper-layer data carried by the datagram. The value is ICMP 1, TCP 6, and UDP 17.

Header Checksum: a 16-bit Header Checksum. This field checksums only the Header of a datagram. Verify whether the IP datagram header is damaged, tampered with, or lost.

Source IP Address: 32-bit Source Address, that is, the IP Address of the sender.

Destination IP Address: 32-bit Destination Address, that is, the IP Address of the receiving end.

### UDP Layer:

Table 2-2 describes the UDP data formats.

Table 2-2: Format of UDP frame headers

|  |  |
| --- | --- |
| Source Port | Destination Port |
| UDP Length | UDP Checksum |
| Data | |

The UDP header consists of 8 bytes and is in the unit of 32 bits (4 bytes), just like the IP header.

Source port number: 16-bit sending port Number. The port number is used to distinguish different services. The port number ranges from 0 to 65535.

Destination port number: 16-bit receiving port number.

UDP length: indicates the 16-bit UDP length, including the length of the UDP header and the length of the data, in bytes.

UDP checksum: 16-bit UDP checksum.

# 3. System design

### 3.1 System Function Description

The system generates data through the counter in FPGA and sends data through the Ethernet module to generate MAC, IP and UDP frame headers, which constitute Ethernet frames. Then transmit the data to FPGA through Ethernet Subsystem IP and connect the two GTY interfaces through 40G optical interface to form a loop. Then transmit the data back to FPGA, pass the Ethernet receiving module, check the Ethernet frame head and output the data conforming to the Ethernet frame. The system block diagram is shown in Figure 3-1:

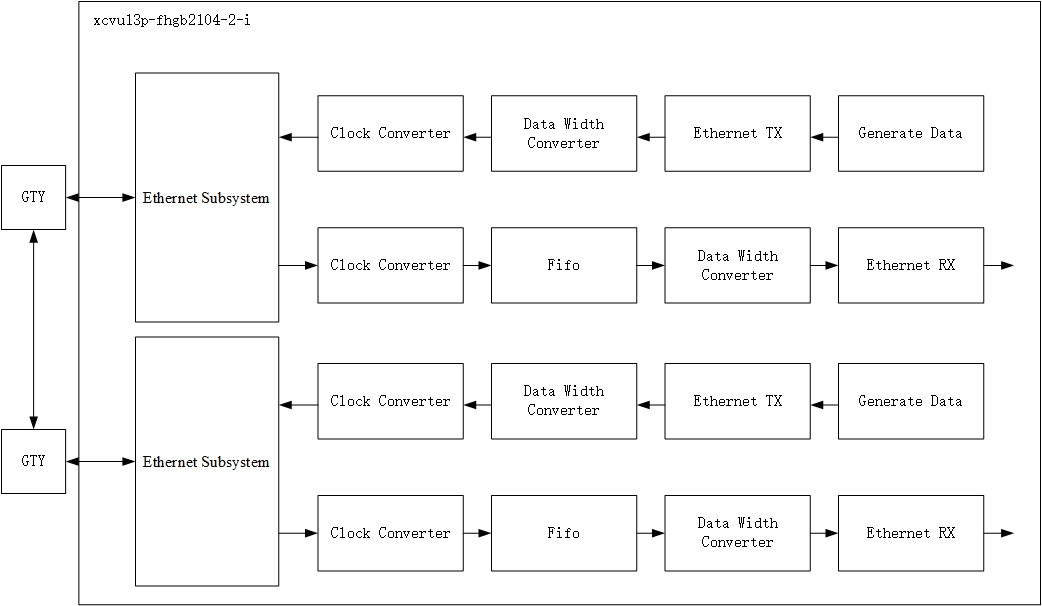


Figure 3-1: The system block diagram

### 3.2 System module division

Ethernet Subsystem IP module: Connects the PHY chip externally through the standard MII interface. When receiving, the MII 4-bit data line converted to 256-bit data line into the internal switch module, and the use of SOP, EOP and MOD control information to identify the starting and effective bits of data frame transmission, Ethernet Subsystem IP can also carry out simple flow control, in addition, an important function is CRC calibration, The Ethernet Subsystem IP core will not pass the CRC data frame directly discarded; When sending, the Ethernet Subsystem IP core receives the data frame from the FPGA, converts the 256-bit data line to the 4-bit data line of the MII interface, and adds the frame checksum (FCS) for the data frame.

Clock Converter IP module: Because the data generated by the Ethernet Subsystem module is driven by the Clock generated by THE PLL on the GTY, so to process the data across the Clock domain, the Clock on both sides of the module is asynchronous 312.5MHz.

Data Width Converter IP module: Since the Ethernet Subsystem IP module generates 256 bits of Data, while the Ethernet TX/RX is 512 bits of Data bit Width suitable for 100GHz, we need to convert the bit Width.

Ethernet RX module: used to parse the input data packet, identify the packet contained MAC address, IP address, UDP port number and protocol type Ethernet frame header information, judge is matched with FPGA information, if matched will continue to transmit data, not in the record for Ethernet frame transmission error.

Ethernet TX module: Used to generate Ethernet frame header for data when there is data input, assemble data into Ethernet frame and output it.

Generate Data module: Generates test Data with a counter to test the correctness of the system.

### 3.3 Overview of Ethernet Transmission module:

### 3.3.1 Ethernet TX module:

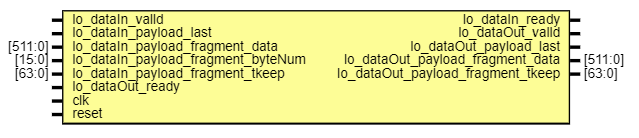
Lists of class:

|  |  |  |  |
| --- | --- | --- | --- |
| Class Name：EthernetTxGenerics | | | |
| Variable Name | Type | Width | Description |
| destIp | Long | 32 | Destination IP Address |
| destMac | Long | 48 | Destination MAC Address |
| destPort | Int | 16 | Destination Port |
| srcIp | Long | 32 | Source IP Address |
| srcMac | Long | 48 | Source MAC Address |
| srcPort | Int | 16 | Source Port |

|  |  |  |  |
| --- | --- | --- | --- |
| Class Name：EthernetTxDataIn | | | |
| Variable Name | Type | Width | Description |
| data | Bits | 512 | Data input into Ethernet TX module |
| byteNum | UInt | 16 | Number of bytes transmitted per packet |
| tkeep | Bits | 64 | The byte qualifier that indicates whether the content of the associated byte of data is processed as part of the data stream |

|  |  |  |  |
| --- | --- | --- | --- |
| Class Name：EthernetTxDataOut | | | |
| Variable Name | Type | Width | Description |
| data | Bits | 512 | Data output from Ethernet TX module |
| tkeep | Bits | 64 | The byte qualifier that indicates whether the content of the associated byte of data is processed as part of the data stream |

The value in EthernetTxGenerics can be configured according to the specific address. The EthernetTxDataIn and EthernetTxDataOut port signals are packaged with the Stream data class of Spinal HDL. Module signal diagram of Verilog code generated at last:



|  |
| --- |
|  |
| **Port name** | **Direction** | **Type** | **Description** |
| io\_dataIn\_valid | input |  | Valid data input. Data transfer is complete only when this signal is 1. |
| io\_dataIn\_ready | output |  | Input confirmation signal, used to indicate the beginning of data transmission |
| io\_dataIn\_payload\_last | input |  | Indicates that the input Ethernet packet is complete |
| io\_dataIn\_payload  \_fragment\_data | input | [511:0] | Data input into Ethernet TX module |
| io\_dataIn\_payload  \_fragment\_byteNum | input | [15:0] | Number of bytes transmitted per packet |
| io\_dataIn\_payload  \_fragment\_tkeep | input | [63:0] | The byte qualifier that indicates whether the content of the associated byte of data is processed as part of the data stream |
| io\_dataOut\_valid | output |  | Valid data output. Data transfer is complete only when this signal is 1. |
| io\_dataOut\_ready | input |  | The output confirmation signal is used to indicate the start of data transmission |
| io\_dataOut\_payload\_last | output |  | Indicates that the output Ethernet packet is complete |
| io\_dataOut\_payload  \_fragment\_data | output | [511:0] | Data output from Ethernet TX module |
| io\_dataOut\_payload  \_fragment\_tkeep | output | [63:0] | The byte qualifier that indicates whether the content of the associated byte of data is processed as part of the data stream |
| clk | input |  | The clock signal |
| reset | input |  | Reset signal of TX module |

Figure 3-2 shows the status transition diagram of the module.

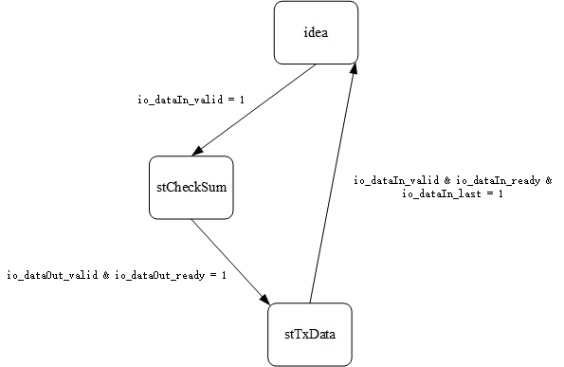


Figure 3-2 : The status transition diagram of Ethernet TX module.

In the IDEA state, wait until the input data valid signal io\_dataIn\_valid is 1. When it is 1, the headBuffer is assigned, the Ethernet frame header data is stored in the headBuffer, and identificationCnt is added by one. In the next clock period, the system jumps to stCheckSum, calculates the IP header checksum, and raises io\_dataOut\_valid to 1. When io\_dataOut\_ready is 1, the system jumps to stTxData. In the stTxData state, the input Stream data Stream is directly output. When the io\_dataIn\_last signal is 1, it indicates that data packet transmission is complete and the data packet jumps back to idea state.

At the same time, the module can also self-configure the MAC, IP, UDP frame header and signal bit width, by changing the constant value of object ErhernetRxConstant, you can configure the module.

|  |  |
| --- | --- |
| **Object Name: ErhernetRxConstant** | |
| **Constant Name** | **Description** |
| DATA\_WIDTH | Data width |
| DES\_IP\_WIDTH | Destination IP width |
| IP\_HEAD\_WIDTH | IP head width |
| ERROR\_CNT\_WIDTH | Error counter width |
| KEEP\_WIDTH | tkeep signal width |
| BYTE\_NUM\_WIDTH | byteNUM signal width |
| PORT\_WIDTH | UDP port width |
| PREAMBLE\_SDF | Preamble and start frame delimiter |
| PREAMBLE\_SDF\_WIDTH | Preamble and start frame delimiter signal width |
| PROTOCOL | Protocol type |
| PROTOCOL\_WIDTH | Protocol type signal width |
| ETH\_TYPE | Ethernet frame type |
| ETH\_TYPE\_WIDTH | Ethernet frame type signal width |
| DEST\_MAC\_WIDTH | Destination MAC width |

### 3.3.2 Ethernet RX module:

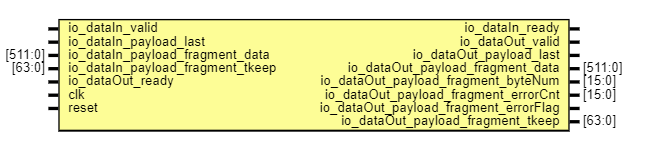
Lists of class:

|  |  |  |  |
| --- | --- | --- | --- |
| Class Name：EthernetRxGenerics | | | |
| Variable Name | Type | Width | Description |
| destIp | Long | 32 | Destination IP Address |
| destMac | Long | 48 | Destination MAC Address |
| destPort | Int | 16 | Destination Port |

|  |  |  |  |
| --- | --- | --- | --- |
| Class Name：EthernetTxDataIn | | | |
| Variable Name | Type | Width | Description |
| data | Bits | 512 | Data input into Ethernet TX module |
| tkeep | Bits | 64 | The byte qualifier that indicates whether the content of the associated byte of data is processed as part of the data stream |

|  |  |  |  |
| --- | --- | --- | --- |
| Class Name：EthernetTxDataOut | | | |
| Variable Name | Type | Width | Description |
| data | Bits | 512 | Data output from Ethernet TX module |
| byteNum | UInt | 16 | Number of bytes transmitted per packet |
| errorCnt | UInt | 16 | Count the number of error packets |
| errorFlag | Bool | 1 | Packet error flag signal |
| tkeep | Bits | 64 | The byte qualifier that indicates whether the content of the associated byte of data is processed as part of the data stream |

The EthernetRxGenerics value can be configured according to the specific address. EthernetTxDataIn and EthernetTxDataOut port signals are packaged with the Stream data class of Spinal HDL. Module signal diagram of Verilog code generated at last:



|  |
| --- |
|  |
| **Port name** | **Direction** | **Type** | **Description** |
| io\_dataIn\_valid | input |  | Valid data input. Data transfer is complete only when this signal is 1. |
| io\_dataIn\_ready | output |  | Input confirmation signal, used to indicate the beginning of data transmission |
| io\_dataIn\_payload\_last | input |  | Indicates that the input Ethernet packet is complete |
| io\_dataIn\_payload  \_fragment\_data | input | [511:0] | Data input into Ethernet RX module |
| io\_dataIn\_payload  \_fragment\_tkeep | input | [63:0] | The byte qualifier that indicates whether the content of the associated byte of data is processed as part of the data stream |
| io\_dataOut\_valid | output |  | Valid data output. Data transfer is complete only when this signal is 1. |
| io\_dataOut\_ready | input |  | The output confirmation signal is used to indicate the start of data transmission |
| io\_dataOut\_payload\_last | output |  | Indicates that the output Ethernet packet is complete |
| io\_dataOut\_payload  \_fragment\_data | output | [511:0] | Data output from Ethernet RX module |
| io\_dataOut\_payload  \_fragment\_byteNum | output | [15:0] | Number of bytes transmitted per packet |
| io\_dataOut\_payload  \_fragment\_errorCnt | output | [15:0] | Count the number of error packets |
| io\_dataOut\_payload  \_fragment\_errorFlag | output |  | Packet error flag signal |
| io\_dataOut\_payload  \_fragment\_tkeep | output | [63:0] | The byte qualifier that indicates whether the content of the associated byte of data is processed as part of the data stream |
| clk | input |  | The clock signal |
| reset | input |  | Reset signal of RX module |

Figure 3-3 shows the status transition diagram of the module.

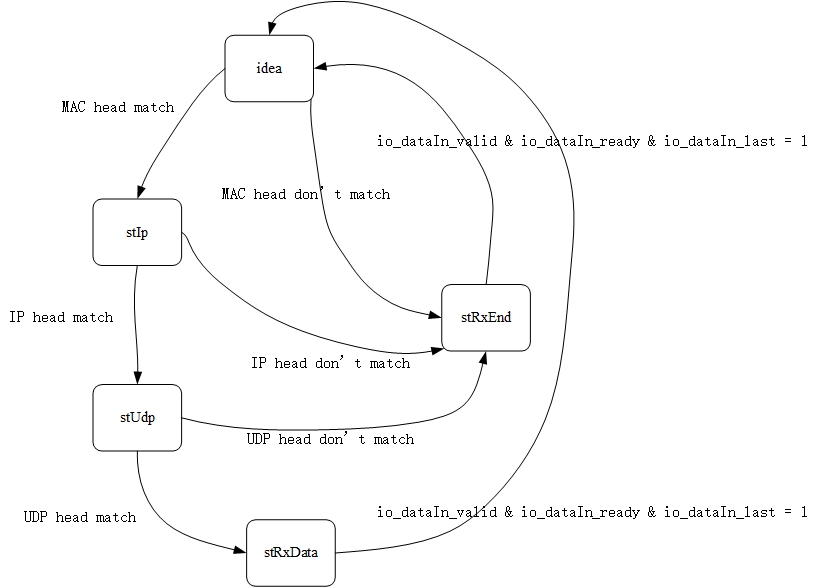


Figure 3-2 : The status transition diagram of Ethernet RX module.

In the IDEA state, wait until the input data valid signal io\_dataIn\_valid is 1 and io.datain.tkeep [63:14] is 50 'H3FFFFFFFFFF. When the data is valid, Start Frame Delimiter for the leading code of the MAC Frame header, Compares the destination MAC address with the protocol type. If match, stIp will be entered, otherwise stRxEnd will be entered, and the errorCnt signal will be accumulated by 1.

In state stIp, IP frame headers are matched and compared. If the protocol type is UDP and the destination IP address matches, state stUdp is entered; otherwise, state stRxEnd is entered, and errorCnt signal is accumulated by 1.

In state stUdp, the destination port number of UDP frame header is matched and compared. If the port number matches, stRxData state is entered; otherwise, stRxEnd state is entered, and errorCnt signal is accumulated by 1.

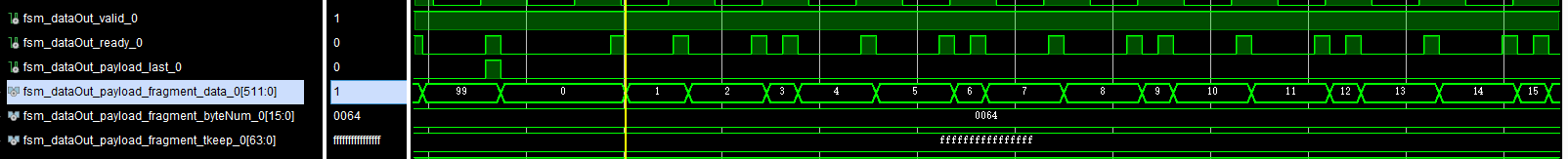
In the stRxData state, the input Stream data Stream is directly output. When the io\_dataIn\_last signal is 1, it indicates that data packet transmission is complete and the data packet jumps back to idea state.

At the same time, the module can also self-configure the MAC, IP, UDP frame header and signal bit width, by changing the constant value of object ErhernetRxConstant, you can configure the module.

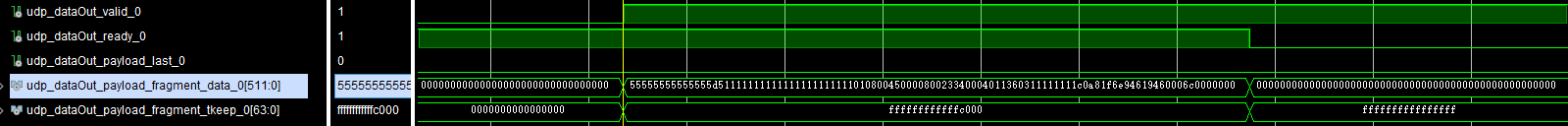
|  |  |
| --- | --- |
| **Object Name: ErhernetRxConstant** | |
| **Constant Name** | **Description** |
| DATA\_WIDTH | Data width |
| DES\_IP\_WIDTH | Destination IP width |
| IP\_HEAD\_WIDTH | IP head width |
| ERROR\_CNT\_WIDTH | Error counter width |
| KEEP\_WIDTH | tkeep signal width |
| BYTE\_NUM\_WIDTH | byteNUM signal width |
| PORT\_WIDTH | UDP port width |
| PREAMBLE\_SDF | Preamble and start frame delimiter |
| PREAMBLE\_SDF\_WIDTH | Preamble and start frame delimiter signal width |
| PROTOCOL | Protocol type |
| PROTOCOL\_WIDTH | Protocol type signal width |
| ETH\_TYPE | Ethernet frame type |
| ETH\_TYPE\_WIDTH | Ethernet frame type signal width |
| DEST\_MAC\_WIDTH | Destination MAC width |

# 4. Simulation and FPGA verification

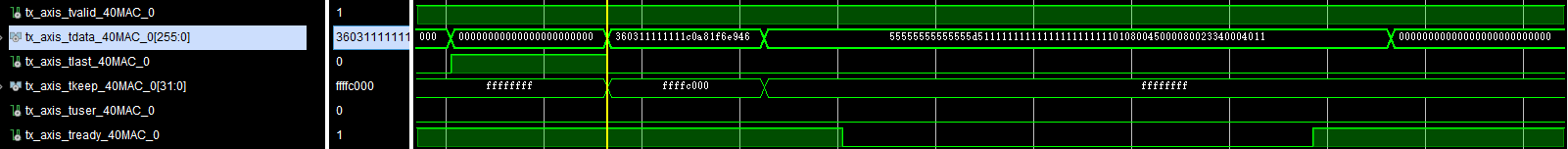
After the system is simulated, the data generated by the data generation module fsm\_dataOut\_payload\_fragment\_data\_0 increases from 0 to 100.



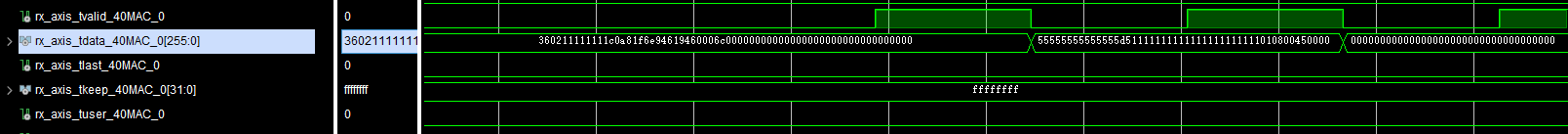
This data is sent to the Ethernet TX module, which generates an Ethernet frame header for the packet:



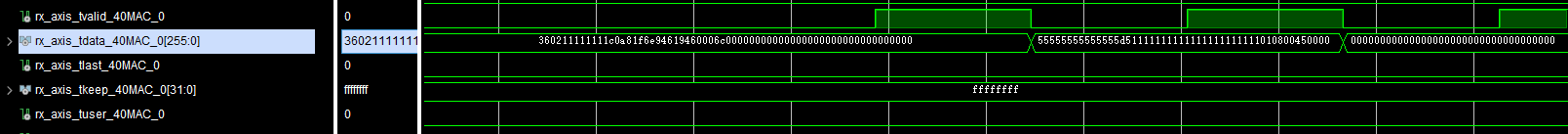
Then through bit width conversion and cross-clock domain module, the 512bit data is converted to 256bit data and fed to the Ethernet Subsystem IP.



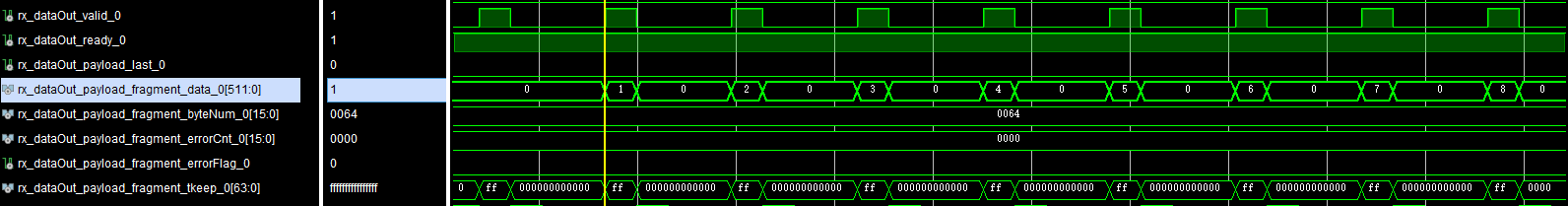
After the Ethernet Subsystem IP is successfully initialized, the two GTY connected by 40G optical module will transmit data to each other to form a loop and receive the data back to FPGA.



The recovered data is then passed through the cross-clock domain module, and the data is stored in The FIFO for subsequent modules to carry out reverse pressure on the received data. Then through the bit width conversion module, the data is converted from 256bit to 512bit.



Finally, the data is sent to the Ethernet RX module, which will check the MAC, IP and UDP frame headers, whether they match the system, if they match, the data will be sent to the subsequent module. Otherwise, the data will not be sent to subsequent modules and the errorFlag signal indicates that the frame header does not match.



Finally, the project was verified by FPGA, and it could be seen that the Ethernet Subsystem IP received data successfully, which constituted a loop:

