Specification of Network Interface Controller

Akshat Ramachandran, Anjana Singh, Harshad Ugale, Madhav P. Desai

1 Introduction

A custom NIC will be designed and interfaced with a 10Gbps MAC so as to provide networking capabilities to the AJIT processor. The NIC consists of 2 threads - the receiver thread and the transmitter thread and these threads operate independently of each other. In iteration 0 , the NIC parses and stores the entire packet in the processor memory (receiver thread), and transmits that packet when instructed by the processor to do so (transmitter thread). The NIC does minimal processing on the packet in this iteration and rests the onus on the host processor. The figure 1 shows full block diagram of the system.

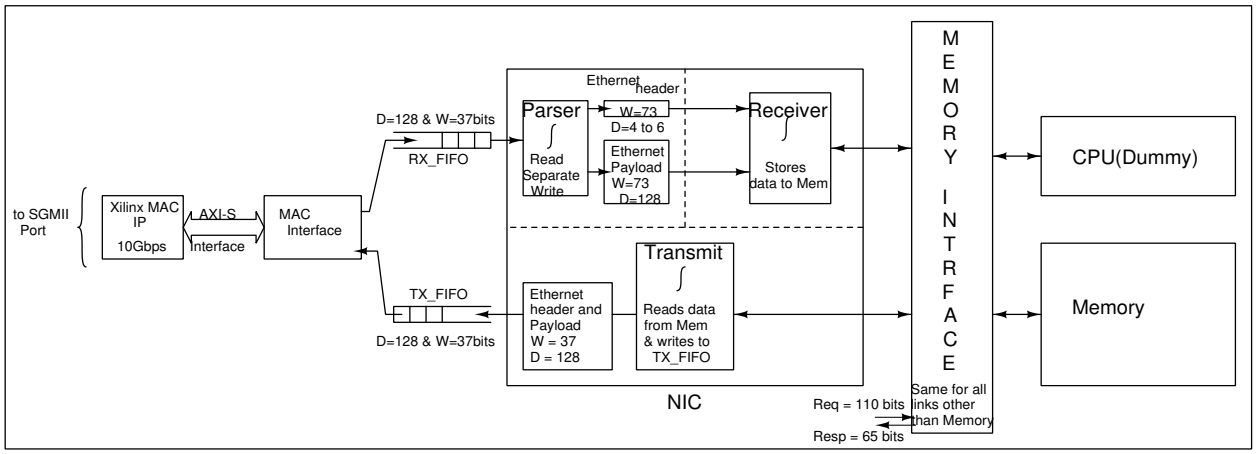

Figure 1: System Architecture

2 MAC FIFOs

The MAC is connected to two FIFOs on the host side namely the Tx_FIFO and the Rx_FIFO. The Rx_FIFO is responsible for the buffering of data sent from the MAC so that the NIC can access the packet at a slightly leisurely pace. The Tx_FIFO buffers the packet to be transmitted by the MAC until the MAC is ready to transmit. The communication between the NIC and the FIFO occurs via the AHIR pipe protocol. The pipe here is defined to be a 37-bit wide pipe and 128 deep, that consists of all the data sent by the MAC on the AXI-s side of the interface or alternatively all the data to be sent to the MAC by the host. The format of the 37-bit data quantity on the pipe is as follows :

tlast	tdata	tkeep
1 bit	32 bits	4 bits

2.1 Error Detection

The MAC does not discard the erroneous packets but instead asserts a signal (tuser) while transmitting the last word of a bad packet. The RX_FIFO notifies the MAC that a bad packet has been received via the 37-bit data word defined earlier, consisting of a unique data pattern (which identifies a bad packet) :

tlast	tdata	tkeep
0b1	0xFFFFFFFF	0x0

When the NIC receives this bad packet identifier, it does not set the flag buffer bit (defined later) and therefore the bad packet is not processed by the host.

3 Parser

Parser pulls the 37 bit data from *RX_FIFO*. Concatenates two chunks and pushes header to *Ethernet_header* pipe and payload to *Ethernet_Payload* pipe.

3.1 Interfaces

- Input : *RX_FIFO* (37 bit wide pipe),

tlast	tdata	tkeep
1 bit	32 bits	4 bits

- Output : *Ethernet_header* and *Ethernet_Payload* (73 bit wide pipes)

tlast	tdata	tkeep
1 bit	64 bits	8 bits

Here at output, tlast(output) is the tlast of last valid chunk(input) and tkeep(output) is concatenated tkeep of two chunks.

4 Receiver Engine

The receiver engine acts as a communication link between memory and the NIC. It receives 73-bit data from the Parser and goes through the following states to store the data in Memory :
Receive Engine Interfaces :

- Input : 73-bit data from Parser consisting of data-words, byte masks and last word identifier,
- Output : 110-bit Request and 65-bit Response (To Memory)

The Receiver Engine polls the control word of each buffer and when an empty buffer is found, it receives the payload and the header from the parser and saves it in the corresponding buffers. And while the last word is being buffered, it determines if the buffered packet is a good or a bad packet (i.e erroneous packet). If the packet is found to be bad, the receiver engine overwrites the same buffer with the next packet. If the packet is good, then the receiver engine sets the flag bit and goes on to find the next empty buffer.

5 Memory Data Storing Format

5.1 Request and Response Format

```
1. Request packet:
   A 110-bit format.

   Request[109:0]
        Request[109]  = lock-bit
        Request[108]  = read/write-bar
        Request[107:100] = byte mask
            write data is organized as 8-bytes. if bit 7 (MSB) of byte-mask
            is set, the top byte of write data is written, else not.
        Request[99:64] = address (36-bit)
        Request[63:0]  = write-data.
```

```
2. Response packet:
   A 65-bit format.
        Response[64] = Error
        Response[63:0] = read-data


3. For every request packet, there is a response.
```

5.2 Addressing Format

The NIC will be reading and writing to 2 buffers-

- The Data Buffer
- The Control Buffer

The buffers will be organised as a 2-Dimensional matrix (16 x 1520) i.e there are 16 buffers with each buffer having a capacity of 1520 bytes. To index into a buffer , 2 indices are needed x,y where, the x index selects the row/buffer and the y index selects the byte chunk within the selected x buffer.
Since the memory has only linear addressing and the 2-D matrix is just an abstraction, there has to be a mechanism to convert from the x,y indices to the actual address in memory which will be done as follows :

For the **Data Buffer** :
Address = offset$_{buffer} + x * s_x + y * s_y$, where $s_x = ((1518 * 8)/64) * s_y \approx 1520$ and $s_y = 8$

For the **Control Buffer** :
Address = offset$_{flag}$ + $x * s_x$, where $s_x = 8$

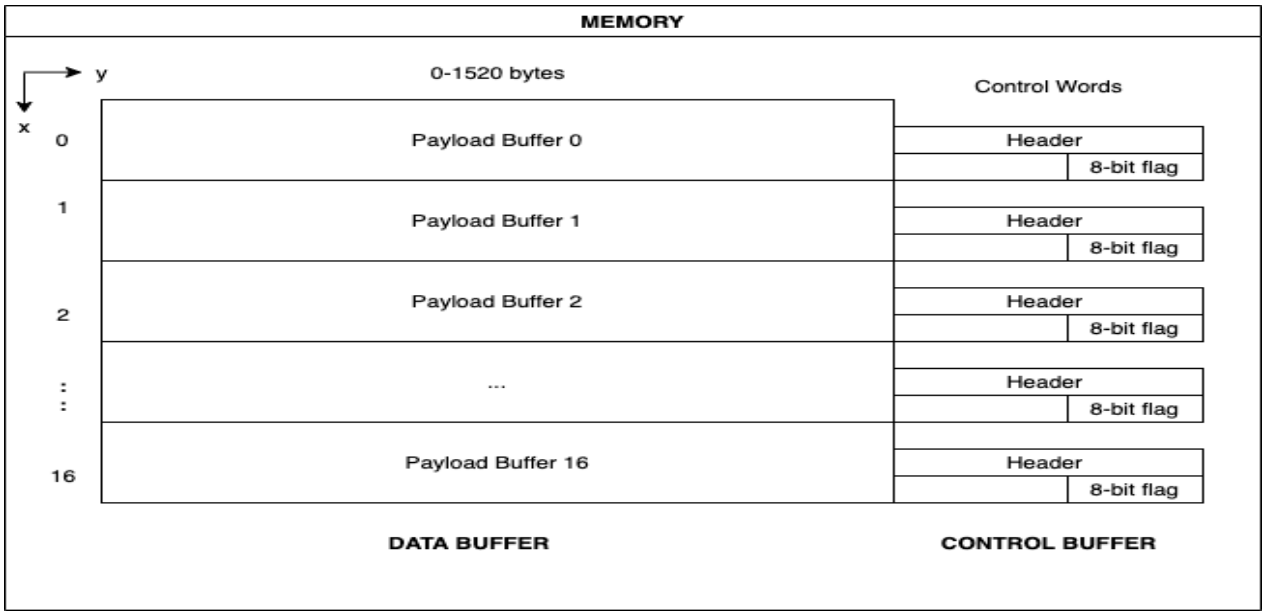

Figure 2: Buffer Organization in Memory

5.3 Control Buffer Format

The control buffer stores the control information including the flag for each buffer in memory i.e if there are n data buffers, there will n corresponding flag buffers.

The 64-bit control word format in the flag buffer is :

Not used	size_of_packet	byte_mask	flag
36 bits	12 bits	8 bits	8 bits

4-bit flag format :

- 00000000 → Buffer empty
- 10000000 → Data written to buffer but not yet processed
- 11010000 → Data processed by processor and ready to transmit
- 11100000 → Data processed, but packet will not be transmitted (drop packet)

6 Transmit Engine

The transmit engine reads data stored in memory (processed by the CPU) and transmits it to the Tx_FIFO. Data is read in 64-bit words and sent out in the 37-bit format described above.

- Input : 65-bit response packet (read data from memory)
- Output: 110-bit request packet and 37-bit data to Tx_FIFO

The engine polls the control buffer (every 32 clock cycles) to check the status of the packet.
It reads the control word and determines if the packet is processed by the CPU, by extracting the flag bits.
If the DROP bit is high, data is not read and the engine moves to the next buffer.
If XMIT bit is HIGH, it starts reading from the data buffer.
The response received is a 65 bit word, with the MSB bit for error. The error bit is checked and if error free, we continue reading till we reach the end of packet. If an error occurs, we send a read request again.

The 64 bit read_data is sliced into 2 chunks of 32 bit each and sent to the Tx_FIFO in the 37-bit format.

After reading (and transmission) is completed, the flag bits are reset and the next buffer is checked.

7 Testing Methodology

For the purpose of verifying the design a simple loopback will be implemented at either the PHY end of the MAC or at the host processor end.

- PHY end : In this approach a packet to be transmitted will be sent by the host processor which will be transmitted by the transmitter thread and eventually is serialised into the txp and txn signals of the SGMII protocol. These signals will be directly connected to the rxp and rxn ports of the MAC and the same packet re-enters the NIC.
- Host processor end : A similar strategy as above will be applied here too, but the point of loopback and the point of packet entry is reversed. Here, the packet will be received by the Mac along the rxp and rxn ports and will be processed by the receiver engine and will be stored in memory. The host processor will process the packet and swap the destination and source addresses in the packet to emulate a loopback.

7.1 Components of the test bench setup

- A module that will be interfaced with the SGMII ports of the MAC and will be monitoring the signals being transmitted and capable of exciting the rxp and rxn ports to emulate the NIC receiving data.
- A memory block that will communicate with the NIC via the request and response protocol and will be storing the packet data and flags in the buffer. Additionally this module will also act as a CPU that will mimic processing being done by the CPU.