

## How to use Multicasting

### Document History

Ver 1.0 (OCT 26, 2005)	First release for W3150A
Ver 2.0 (AUG 15, 2006)	Second release for W3150A+ Added figure and more information

© 2006 WIZnet Co., Inc. All Rights Reserved.

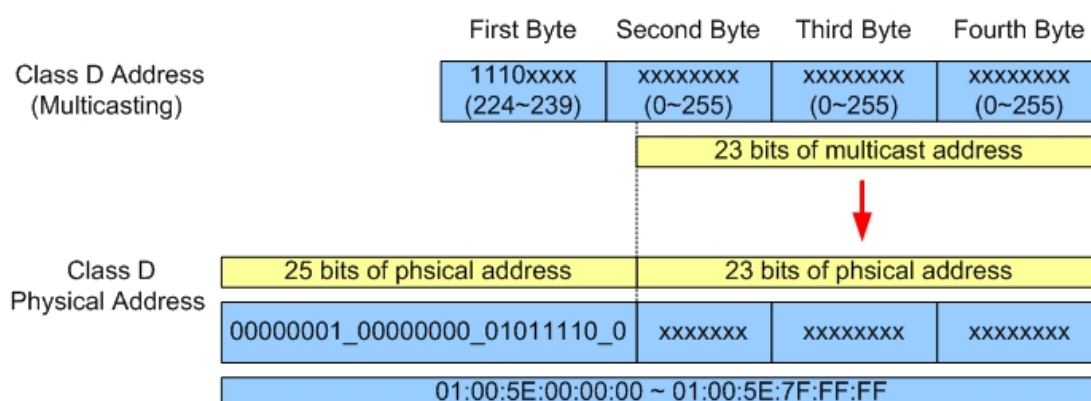
For more information, visit our website at <http://www.wiznet.co.kr>

## IP Multicasting

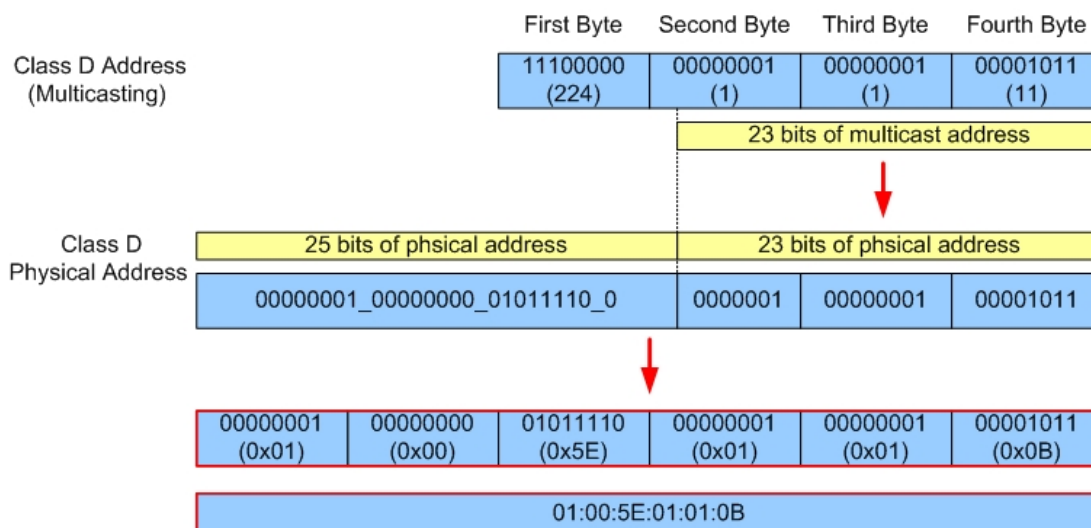
IP Multicasting provides delivery to multiple destinations belonging to same multicast group.

Multicast group addresses are in the range of 224.0.0.0 through 239.255.255.255. (CLASS D), and multicast hardware(Ethernet MAC) address corresponding to IP multicasting address are in the range of 01:00:5e:00:00:00 through 01:00:5e:7f:ff:ff. This allocation allows for 23 bits in the Ethernet MAC to correspond to the IP multicast group address. The lower 23 bits of the multicast group address is placed into the 23 bits of the Ethernet MAC.

Below figure shows Multicasting Address and Multicasting Hardware Address.



For example, assume that multicast group address is 224.1.1.11 and multicast group port number is 3000. In this case, multicast Hardware address is 01:00:5e:01:01:0b.



### 1. Phase 0

### - IGMP socket initialization

Before Multicasting communication, each destination sends IGMP(Internet Group Management Protocol) message to the gateway in order to join Multicasting Group.

Before set OPEN command on W3150A+ socket N command register, write the Multicasting group address, Multicasting Hardware Address and port number to SO\_DIPR(Destination IP Register), SO\_DHAR(Destination Hardware Address Register) and SO\_DPORT(Destination Port Number Register).

```
{
    /* Write Multicasting Group Address, Multicasting Hardware Address and port number on
    related registers in the socket that selected for multicasting communication. */

    /* example code uses socket 0 in multicasting conditions */
    SO_DIPR   = 0xE001010B;    // e.g., 224.1.1.11
    SO_DHAR   = 0x01005e01010B; // e.g., 01.00.5e.01.01.0b
    SO_DPORT  = 0x0BB8;        // e.g., 3000

    /* set UDP, Multicasting on socket 0 mode register */
    SO_MR     = 0x02 | 0x80;

    /* set OPEN command */
    SO_CR = OPEN;
}
```

## 2. Phase 1

### - Data communication

The multicasting receiving process is same as UDP receiving process.

The only difference is two values of Destination Address field and Destination Hardware Address field in receiving packets have Multicasting Group Address, Multicasting Hardware Address. In normal UDP receiving process, W3150A+ will receive packets when destination information in receiving packets equal to W3150A+'s source information, but in multicasting data communication, destination information in receiving packets is multicasting information. As for W3150A+, the receiving process is same as TCP receiving, because multicast group address and multicast group port number are already assigned as remote address and port number.

### 2.1 Receiving Process

Data receiving process is as below. In case of UDP, 8byte header is attached to the receiving data packet. The structure of the header is as below.

DEST. IP Address (4)	DEST. Port (2)	Data size (2) (*data size except for 8byte of header)
----------------------	----------------	---

Multicasting process does not need the destination IP address and port number which are used for UDP process. Therefore, the process to acquire destination IP address and port number can be removed in the multicasting communication process.

Even though, destination IP address and port number are recorded, W3150A+ will operate appropriately with the multicasting address and port number.

Below is about normal UDP process. By removing the process of acquiring destination IP address and port number, it can be used for the multicasting process.

```

{
    /* first, get the received size */
    get_size = Sn_RX_RSR;
    /* calculate offset address */
    get_offset = Sn_RX_RR & gSn_RX_MASK;
    /* calculate start address(physical address) */
    get_start_address = gSn_RX_BASE + get_offset;

    /* read head information (8 bytes) */
    header_size = 8;
    /* if overflow socket RX memory */
    if ( (get_offset + header_size) > (gSn_RX_MASK + 1) )
    {
        /* copy upper_size bytes of get_start_address to header_addr */
        upper_size = (gSn_RX_MASK + 1) - get_offset;
        memcpy(get_start_address, header_addr, upper_size);
        /* update header_addr */
        header_addr += upper_size;
        /* copy left_size bytes of gSn_RX_BASE to header_addr */
        left_size = header_size - upper_size;
        memcpy(gSn_RX_BASE, header_addr, left_size);
        /* update get_offset */
        get_offset = left_size;
    }
}
    
```

```

}
else
{
    /* copy header_size bytes of get_start_address to header_addr */
    memcpy(get_start_address, header_addr, header_size);
    /* update get_offset */
    get_offset += header_size;
}
/* update get_start_address */
get_start_address = gSn_RX_BASE + get_offset;

/* save remote peer information & received data size */
/* peer_ip and peer_port are not needed */
/* peer_ip = header[0 to 3]; removed */
/* peer_port = header[4 to 5]; removed */
get_size = header[6 to 7];

/* if overflow socket RX memory */
if ( (get_offset + get_size) > (gSn_RX_MASK + 1) )
{
    /* copy upper_size bytes of get_start_address to destination_addr */
    upper_size = (gSn_RX_MASK + 1) - get_offset;
    memcpy(get_start_address, destination_addr, upper_size);
    /* update destination_addr */
    destination_addr += upper_size;
    /* copy left_size bytes of gSn_RX_BASE to destination_addr */
    left_size = get_size - upper_size;
    memcpy(gSn_RX_BASE, destination_addr, left_size);
}
else
{
    /* copy get_size bytes of get_start_address to destination_addr */
    memcpy(get_start_address, destination_addr, get_size);
}
/* increase Sn_RX_RR as length of get_size+header_size */
Sn_RX_RR = Sn_RX_RR + get_size + header_size;

```

```

/* set RECV command */
Sn_CR = RECV;
}
    
```

## 2.2 Sending Process

As like receiving process, by removing the process to acquire destination IP address and port number from normal UDP sending process, data transmission is processed in multicasting.

```

{
    /* first, get the free TX memory size */
FREESIZE:
    get_free_size = Sn_TX_FSR;
    if (get_free_size < send_size) goto FREESIZE;

    /* Write the value of remote_ip, remote_port to the Socket n Destination IP Address
       Register(Sn_DIPR), Socket n Destination Port Register(Sn_DPORT). */
    /* peer_ip and peer_port are not needed */
    /* Sn_DIPR = remote_ip; */
    /* Sn_DPORT = remote_port; */

    /* calculate offset address */
    get_offset = Sn_TX_WR & gSn_TX_MASK;
    /* calculate start address(physical address) */
    get_start_address = gSn_TX_BASE + get_offset;

    /* if overflow socket TX memory */
    if ( (get_offset + send_size) > (gSn_TX_MASK + 1) )
    {
        /* copy upper_size bytes of source_addr to get_start_address */
        upper_size = (gSn_TX_MASK + 1) - get_offset;
        memcpy(source_addr, get_start_address, upper_size);
        /* update source_addr */
        source_addr += upper_size;
        /* copy left_size bytes of source_addr to gSn_TX_BASE */
        left_size = send_size - upper_size;
        memcpy(source_addr, gSn_TX_BASE, left_size);
    }
}
    
```

```
}  
else  
{  
    /* copy send_size bytes of source_addr to get_start_address */  
    memcpy(source_addr, get_start_address, send_size);  
}  
/* increase Sn_TX_WR as length of send_size */  
Sn_TX_WR += send_size;  
/* set SEND command */  
Sn_CR = SEND;  
}
```