- Application Note for W5100 -

How to use Multicasting

Document History

Ver 1.0 (JAN 3, 2007)	First release

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

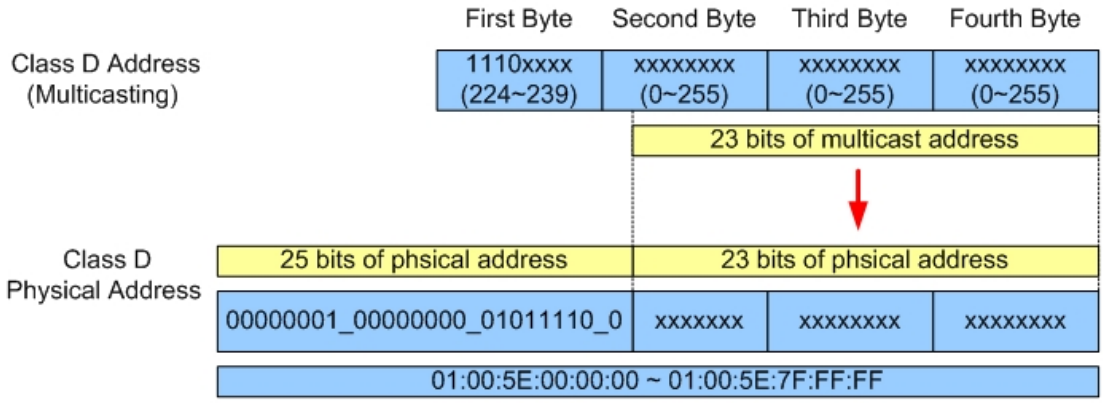
IP Multicasting

IP Multicasting provides delivery to multiple destinations belong to same multicast group. Multicasting Address uses only for Destination Address value.

Multicast group addresses are in the range 224.0.0.0 through 239.255.255.255.(CLASS D)

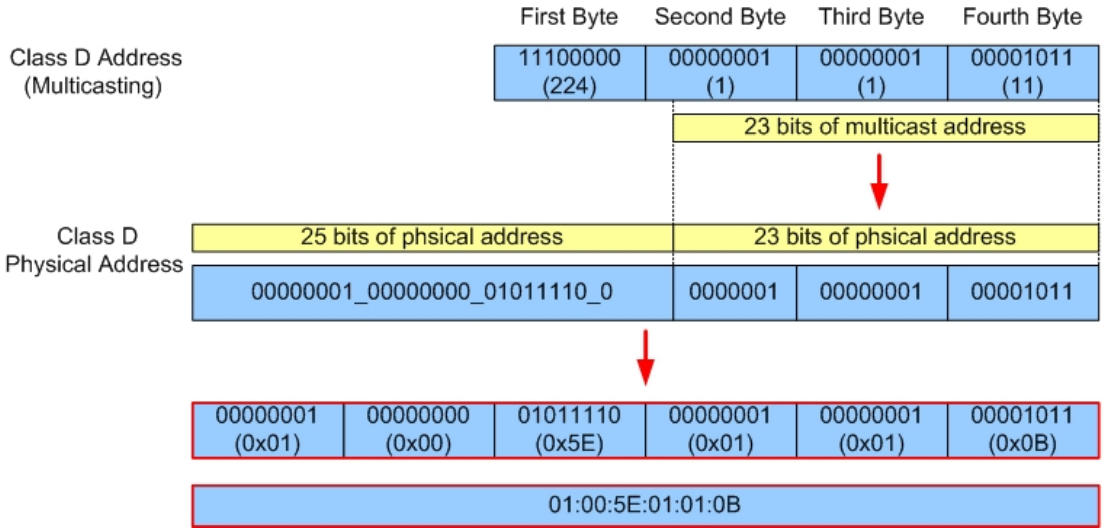
And multicast hardware(Ethernet MAC) address corresponding to IP multicasting address are in the range 01:00:5e:00:00:00 through 01:00:5e:7f:ff:ff. This allocation allows for 23 bits in the Ethernet MAC to correspond to the IP multicast group address. The mapping places **the low-order 23 bits of the multicast group address into these 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.

Below figure shows a example.



1. Phase 0

- IGMP socket initialization

For Multicasting communication, each destinations send IGMP(Internet Group Management Protocol) message to gateway to join Multicasting Group before Multicasting data communication.

Before set OPEN command on W5100 socket N command register, write the Multicasting group address, Multicasting Hardware Address and port number to S0_DIPR(Destination IP Register), S0_DHAR(Destination Hardware Address Register) and S0_DPORT(Destination Port Number Register).

```
{
    /* Write Multicasting Group Address, Multicasting Hardware Address and port number
       at related register in socket that selected for multicasting condition by user. */
    /* example code uses socket 0 in multicasting conditions */
    S0_DIPR   = 0xE001010B;       // e.g., 224.1.1.11
    S0_DHAR   = 0x01005e01010B; // e.g., 01.00.5e.01.01.0b
    S0_DPORT = 0x0BB8;             // e.g., 3000

    /* set UDP, Multicasting on socket 0 mode register */
    S0_MR    = 0x02 | 0x80;

    /* set OPEN command */
    S0_CR = OPEN;
}
```

2. Phase 1

- Data communication

The Multicasting receiving process is same to general UDP receiving process.

The only difference is two values on Destination Address field and Destination Hardware Address field in receiving packets have Multicasting Group Address, Multicasting Hardware Address. In normal UDP receiving process, W5100 will receive packets when destination information in receiving packets equal to W5100's source information. But in multicasting data communication, destination information in receiving packets is multicasting information. In W5100 operation respect, the process of receiving is same to TCP receiving, because remote address and remote port number is already fixed as multicast group address and multicast group port number.

Multicasting sending process is equal to multicasting receiving process and multicasting

sending operation is same to multicasting receiveing operation also.

2.1 Receiving Process

Received data can be processed as below. In case of UDP, 8byte header is attached to receiving data. 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)

Destination ip address and port number are not needed in multicasting process.
So partial of taking destination ip address and port number in UDP process is not needed.
Thus the process of taking destination ip address and port number can be removed in multicasting process.
Even if write destination ip address and port number, W5100 operates normally because W5100 use multicasting address and port number itself.
Below receiving process is normal UDP process. Remove partial process of taking destination ip address and port number, then below process changes to 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

Sending process is equal to receiving process. Remove partial process of taking destination ip address and port number in normal UDP sending process.

Data transmission process is as below.

```
{
    /* 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;
}
```