Application Report
SPRAAI3 – February 2007

Using IP Multicasting with the TMS320C6000 Network Developer's Kit (NDK)

Arnie Reynoso, Rafael de Souza *SDO Applications*

ABSTRACT

This document describes the multicasting support in the TI TMS320C6000 Network Developer's Kit (NDK). It covers the specifics of the NDK API and its differences from the Berkeley Software Distribution (BSD) TCP/IP stack. This document also analyzes example host and client applications that illustrate use of the NDK in a multicasting environment.

This document references example applications that can be downloaded using this link: http://www-s.ti.com/sc/techlit/spraai3.zip.

Contents

1 Overview of Multicasting in IP Networks

The IPv4 version of the Internet Protocol (IP) discusses two types of point-to-multipoint communications:

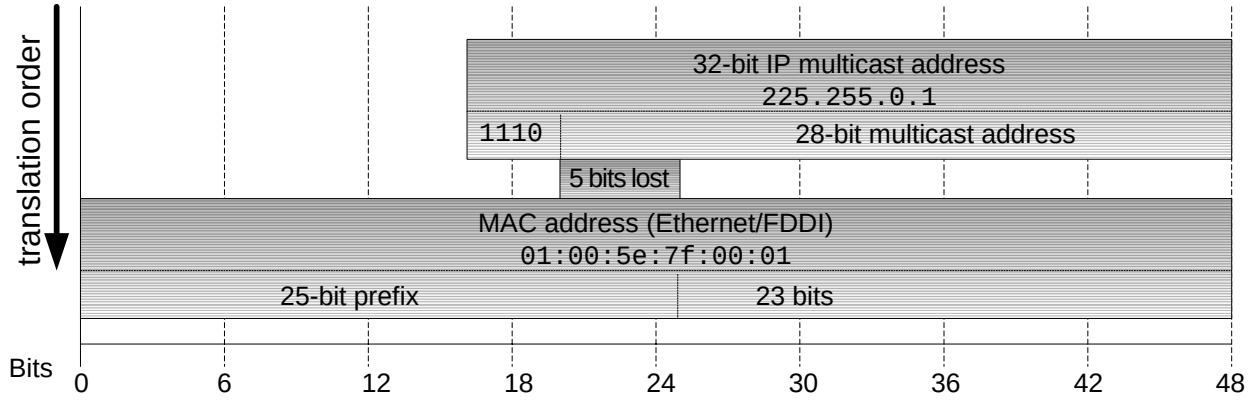
- **Broadcast communications**, in which a host sends information addressed to the entire network. IPv4 requires support for broadcast. (IPv6 drops the broadcast requirement.)
- **Multicast communications**, in which a host sends information to a specific group of hosts. IPv4 makes support for multicast optional.

These two communication methods are enabled by assigning specific address numbers at both the protocol layer (IP) and the link layer (Ethernet). Only IPv4 is supported by the TMS320C6000 Network Developer's Kit (NDK) at this time; the information in this application note refers to the IPv4 version.

1.1 Broadcast vs. Multicast and How They are Supported

For broadcast communications to be able to access all hosts on the network, the destination IP address is the subnet address with all the host ID bits set. For example, if the subnet is 192.168.240.xxx, then the broadcast address is 192.168.240.255—this is called *directed broadcast*. However, the official IP broadcast address has all bits set 255.255.255.255, which is what NDK uses for broadcast communications. Both these addresses are translated to the link layer address (MAC address) as FF:FF:FF:FF:FF:FF, and are received by every Ethernet adapter in the subnet.

Multicast communications are more complex—they require virtual groups of hosts to have an associated unique ID (or address). To differentiate groups from ordinary hosts, the IP standard assigns the address range from 224.0.0.0 to 239.255.255.255 (called *class D* addresses) to hold groups in the subnet. The lower 28 bits form the multicast *group ID.* The full 32-bit address is called the *group address*. Only 23 bits (of the full 28 bits) of the *group ID* are translated to the link layer, followed by a zero in the 24th bit. The remaining 24 bits are always 01:00:5E. See Section 1.3 for information about address assignment and potential issues.



At the transport layer, both broadcast and multicast communications require UDP (User Datagram Protocol) or raw IP sockets to perform communications, since TCP is a connection-oriented protocol designed for use in point-to-point communications.

Despite being mandatory in IPv4, broadcast is losing ground to multicast due to its main disadvantages:

- It is limited to LAN environments
- It creates an overhead on the network, since every host needs to process the incoming data, even those that are not designated recipients.

Multicasting, although it is optional in IPv4, has the advantages of being usable in WAN environments and allowing data transfers to occur selectively in groups of hosts, which minimizes the processing overhead throughout the network. This is a clear advantage when multiple clients reply to a single server in a video stream application, for example.

However, since multicast is an optional feature, all the hosts and routers must be specifically enabled to perform this functionality; fortunately it's a common feature nowadays with one catch—although multicasting can be used in WAN environments, currently only enterprise routers implement this feature, and only using proprietary protocols.

1.2 How Does Multicasting Work?

As mentioned before, multicasting uses the concept of virtual groups and each host in the network must explicitly join this group to receive its data stream.

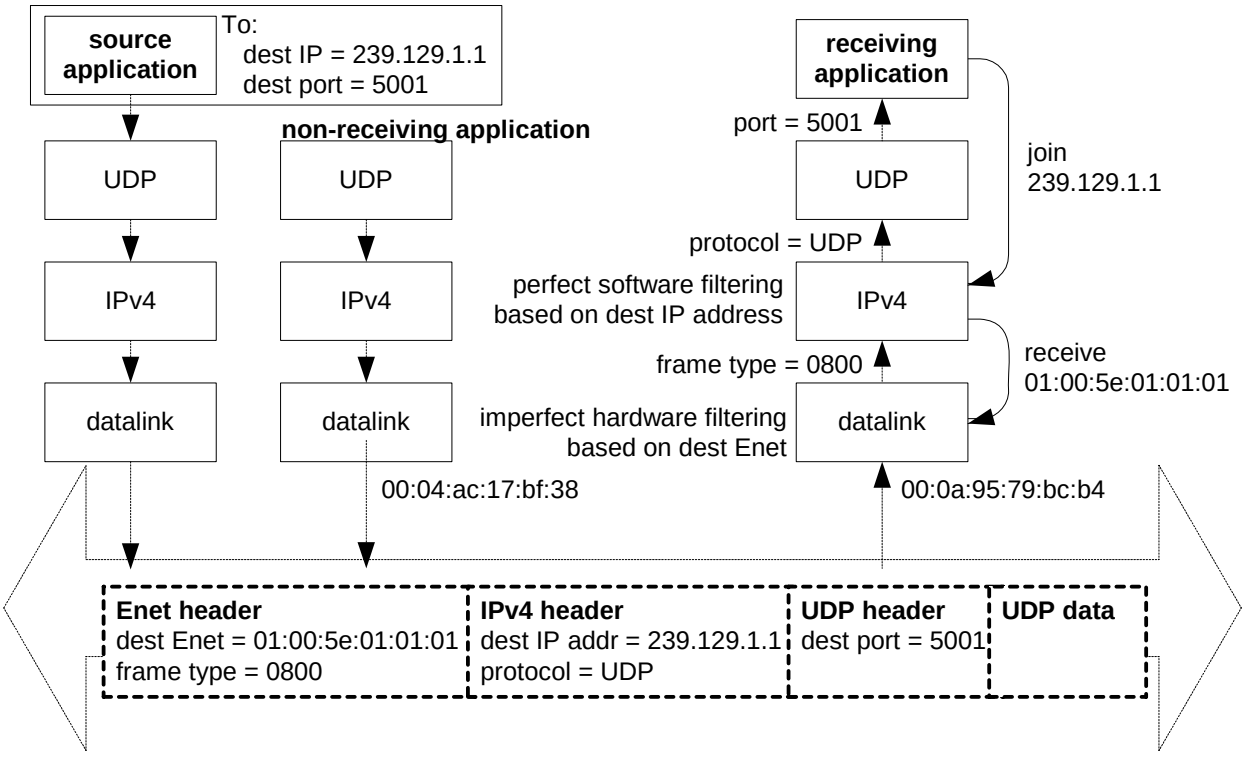
Both the link (Ethernet) and the protocol (IP) layers are responsible for determining the group and selecting the packets that are processed by every host in the network. The Ethernet adapter filters the incoming packets that contain its own MAC address, a broadcast address or, if enabled by the upper layer, a multicast address that starts with 01:00:5E. The protocol layer (IP) in the host then checks to see if the class D address matches the multicast group of interest. Section 1.4 contains more details about the filtering process.

In order to enable a host to receive the multicast addresses, a host management protocol at the subnet level is performed by an IGMP-type IP packet (Internet Group Management Protocol). TI's Network Development Kit (NDK) incorporates release 2 of this protocol (IGMP v2), which has the following types of messages:

- Membership query
- Version 1 membership report
- Version 2 membership report
- Leave group

To join a multicast group, a host sends out a *Version 2 membership report* containing its own IP address as the source address and the *group address* as destination. This instructs the router to route packets to this host when the destination address is the same as the *group address*. Similarly, if a host wants to leave a group, it sends a *Leave group* message. The router periodically listens to IGMP messages and sends *Membership query* messages to check which groups and hosts are still active on its subnet.

At the transport layer, the host must create a UDP socket to receive data. To avoid the problem mentioned earlier of the hosts receiving all the UDP traffic on the network, it's a good idea to bind each host to a specific port. The host then receives all multicast communications for a particular group / port combination.



Similarly, to be able to send data to a multicast group, a host must create a UDP socket with the same port and group address as the listening socket. The same socket cannot be used to send and receive packets, since the IP standard forbids packets from having a multicast address as the source address.

At the network-level, a multicast-enabled *router* manages groups inside its subnet. Every multicast-enabled *host* within that subnet is then able to join groups. The groups can then send data to all hosts in a group. However, this situation does not occur if every host is directly connected to an interface port in the router.

Multicast communications can also be shared among peer routers in a WAN environment by registering groups to the other routers through a Multicast Routing Protocol (MRP). However, as mentioned before, currently only enterprise routers implement this feature using proprietary protocols; most off-the-shelf, low-cost routers and gateways do not support it.

In summary, to be able to perform multicast communications a system needs to:

- Create the UDP sending socket using the multicast address and the port ID.
- Create the UDP receiving socket and bind it to the same address / port combination.
- Join the multicast group by sending an IGMP IP packet to the multicast address.

Finally, functions to send data and process received data are tied to the sockets. This is heavily dependent on the protocol stack used. See Section 2 for details on how to do it with TI's NDK.

1.3 A Word About Class D Addresses

As mentioned earlier, the IP standard assigns the address range from 224.0.0.0 to 239.255.255.255 to act as class D addresses, which hold groups. However, several addresses are reserved for special purposes by IANA (Internet Assigned Numbers Authority), the organization responsible for this.

IANA assigned the range between 224.0.0.0 through 224.0.0.255 for network and routing management purposes. These addresses are neither sent to other routers, nor can they be assigned to user-created multicast group IDs. This range is called Reserved Link Local Addresses; some of the most well-known assignments are listed here:

- 224.0.0.1 – all systems on the subnet
- 224.0.0.2 – all routers on the subnet
- 224.0.0.5 – OSPF routers (Open Shortest Path First)
- 224.0.0.6 – OSPF designated routers (Open Shortest Path First)
- 224.0.0.12 – DHCP server/relay agent

The range between 224.0.1.0 and 238.255.255.255 is designated for WAN-enabled multicast group addresses, called “globally scoped addresses”. The IANA has already assigned some addresses in this range to special services or to equipment vendors. Despite being ordinary multicast group IDs, it is a good practice not to use IANA-assigned addresses in order to guarantee compatibility among networks.

The last address range is called “limited” or “administratively scoped addresses” (subnet only). These are in the range from 239.0.0.0 to 239.255.255.255. These addresses are limited to subnet scope and can also be reused among routers within the same domain.

The IANA website at www.iana.org has a complete table of assigned class D addresses.

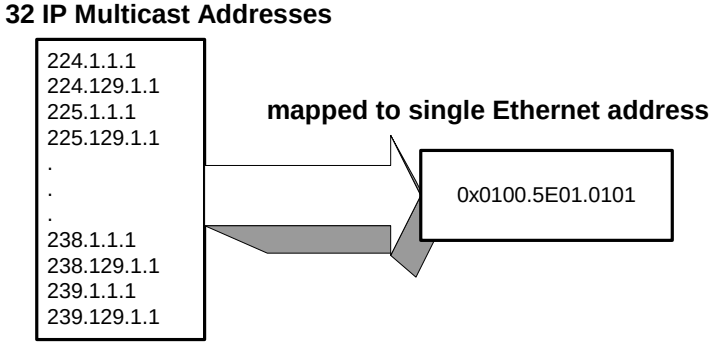
1.4 A Word About Multicast Filtering

To minimize the overload for non-assigned recipients, modern Ethernet interface cards use multicast filtering to receive only messages to multicast addresses assigned to its own host at the link layer level. The filter is set when the host joins a multicast group.

Internally, a hash function is applied to the destination multicast address to calculate an arbitrary value between 0 and 511 and to set its bit to one. When a multicast frame reaches the Ethernet adapter, it applies the same hash function to the destination address contained in the frame and sends it to the next layer (IP layer) if the previously assigned bit is set.

This is all done at link layer and is heavily dependent on the adapter’s filtering length. While 512 bits is able to filter out every non-desirable multicast datagram (called *perfect filtering*), some adapters use only 64 bits or even have no filter at all.

Even when using a perfect filtering adapter, filtering is still needed at the IP layer level, since the translation from the IP multicast address to the Ethernet address is not one-to-one—a total of 32 multicast IP addresses are always mapped to the same Ethernet address. In other words, since each multicast group ID is comprised of 28 bits, but 5 bits are dropped by the adapter, 2^5 or 32 different addresses share the same 48-bit MAC address, leaving to the IP layer the task of uniquely identifying each group ID. Support for IP layer filtering started in NDK release 1.92.



2 TCP/IP Stack Multicast Support

This section describes the main differences between the Berkeley Software Distribution (BSD) socket calls and the TCP/IP stack in the TMS320C6000 Network Developer's Kit (NDK). The NDK supports multicasting only from a client perspective.

2.1 BSD vs. NDK TCP/IP Stacks

The most significant difference between the BSD implementation of multicast and the NDK is that the C6000 TCP/IP stack does not associate IGMP actions with a particular socket. This means that all multicast operations apply to the stack itself, not just to the socket in question. Although the BSD API defines multicast operations on a per-socket basis, some implementations are internally global, while others perform socket-specific filtering. Thus behavioral differences between a BSD-derived implementation and the C6000 TCP/IP will vary.

Nine BSD socket options support multicasting. Of these, the NDK supports four: two that affect sending UDP datagrams to a multicast address and two that affect receiving multicast datagrams. The NDK does not directly support these through socket options, but accomplishes this through other APIs. The following list translates BSD socket options to NDK APIs with comments on differences:

- **IP_ADD_MEMBERSHIP → IGMPJoinHostGroup()**
  The stack must have a valid IP address before calling IGMPJoinHostGroup().
- **IP_DROP_MEMBERSHIP → IGMPLeaveHostGroup()**
  A local "reference count" is kept when "join" is called. Therefore "leave" must be called as many times as "join" to leave the group.
- **IP_MULTICAST_IF → SO_IFDEVICE**
  By binding the multicast socket to a local IP address, the egress IF is always the IF of the locally-bound address. By binding to a wildcard address, the SO_IFDEVICE will be used.
- **IP_MULTICAST_TTL → IP_TTL**
  This sets the TTL for all packets sent on the socket not just multicast data.

The following BSD socket options are not currently supported by the NDK:

- IP_MULTICAST_LOOP
- IP_BLOCK_SOURCE / IP_UNBLOCK_SOURCE
- IP_ADD_SOURCE_MEMBERSHIP / IP_DROP_SOURCE_MEMBERSHIP

Note: The sendto() command is used to transfer UDP datagrams. They need a minimum data size of 8.

2.2 Limitations

Currently the NDK supports membership to only 32 multicast groups. This limits the number of groups you can tell your multicast router you wish to receive. The limit (32) is also used to program the EMAC hardware; therefore, this limits the number of incoming multicast addresses. There is no limit on the number of multicast addresses that can be sent out.

Not all devices supported by the NDK have perfect filtering. They all use a hash table. The 32 multicast address limit is set by the EMAC used on some of these devices. Even EMAC with perfect filtering would have a fixed limit to the number of addresses.

3 Example Applications

This section describes the example applications provided with this document. One example is a client-side application that receives UDP datagram packets sent to a multicast address. Another example is a simulated “host” application that is responsible for sending UDP packets to the group. The sections that follow detail both examples, pointing out the APIs and socket calls used to support multicasting.

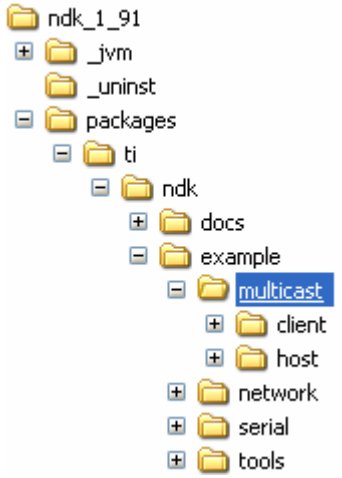
Examples for both the host and client applications are provided on all platforms supported by the NDK. The examples require v1.91 or greater of the NDK to work properly.

3.1 Installing the Examples

The examples are provided in a zip file that can be downloaded using this link: http://www-s.ti.com/sc/techlit/spraai3.zip.

Important: Extract the zip file in the example folder of your NDK folder. If you extract it elsewhere, the example will not build correctly.

After extracting the files, the example folder contains a “multicast” folder with various other sub-folders. It should look similar to the directory tree shown here.



To run the example you need at least two boards supported by the NDK. You also need an off-the-shelf router (with DHCP support) or a network with multicast routing support.

Only one instance of the simulated “host” example application can be running at any given time. As for the client example application, there is no limit to the number of boards running the example (so long as all boards have unique MAC addresses).

3.2 Host Example

The NDK supports multicasting from a client perspective only. This example is a simulated “host” that illustrates how the NDK can be used for multicasting.

The example has a simple UDP (udp_test() function) server task that send a message to the multicast address every few seconds. The UDP server function listens for responses from all clients listening to the same multicast address. The following snippet of code from the example highlights the APIs and socket calls used for multicasting.

```
static void udp_test()
{
...
    // Join the Mulicast Group for MulticastAddr on interface "1"
        if (IGMPJoinHostGroup( inet_addr(MulticastAddr), 1 ))
            printf("Successfully joined the multicast group %s\n", MulticastAddr);
...
    // Receive socket: set Port = 5001, IP address = ANY
    bzero( &sin1, sizeof(struct sockaddr_in) );
    sin1.sin_family = AF_INET;
    sin1.sin_len    = sizeof( sin1 );
    sin1.sin_port   = htons(5001);

    // Send socket: set Port = 5000, IP address = ANY
    bzero( &sout1, sizeof(struct sockaddr_in) );
    sout1.sin_family = AF_INET;
    sout1.sin_len    = sizeof( sout1 );
    sout1.sin_port   = htons(5000);
...
    for(;;) {
         sout1.sin_addr.s_addr = inet_addr(MulticastAddr);

         // Send to multicast group/port every 5 sec until successful
         do {
             TSK_sleep(5000);
             sentCnt = sendto( send, sendBuffer, 15, 0, &sout1, sizeof(sout1) );
         } while (sentCnt < 0);

        printf (" Message sent to multicast group \n");

        // Check for network data on port 5001
        FD_ZERO(&ibits);
        FD_SET(recv, &ibits);
...
        // See if there is data on the UDP socket...
        if( FD_ISSET(recv, &ibits) ) {
            // Read the data
            tmp = sizeof( sin1 );
            cnt = (int)recvfrom( recv, recvBuffer, 1500, 0, &sin1, &tmp );
...
}
```

The first action performed is to “join” a multicast group. It then sets up two sockets to send (port 5000) and receive (port 5001) UDP data. After the setup, a data message is sent to the multicast address using port 5000 every few seconds until a client joins the group and begins receiving the message. The application then listens on port 5001 for a reply from any client that received the multicast message.

3.2.1 Building and Running the Host Example

A Code Composer Studio (CCStudio) project file is provided with the example. The project (host-multicast.pjt) is located in the examples/multicast/host/<*board*> folder. The example is provided as source only; you must re-build it for your particular version of the tools supported by the NDK. After building the example, you can simply load and run the example.

Ensure that the board is connected to either a standalone DHCP-supported network router (preferred) or to a network with multicast routing capability.

The example prints (to the standard out window) a string when a message is sent. Once a multicast message is received, it prints a message indicating the IP address of the board that replied to the original message.

```
TCP/IP Multicast Example Host
Using MAC Address: 00-0e-99-02-71-57
Service Status: DHCPC    : Enabled  :          : 000
Service Status: DHCPC    : Enabled  : Running  : 000
Link Status: 100Mb/s Full Duplex on PHY 1
Network Added: If-1:156.117.95.197
Service Status: DHCPC    : Enabled  : Running  : 017
Successfully joined the multicast group 224.1.2.3
 Message sent to multicast group
Reply #1 from: 156.117.95.162
 Message sent to multicast group
Reply #2 from: 156.117.95.162
```

3.3 Client Example

This example is a daemon client example that works in conjunction with the previously described host example. The example has a simple UDP daemon that listens for a message sent to a multicast address on port 5000. The UDP server function replies to the host when the message is received. The following snippet of code from the example highlights the APIs and socket calls used for multicasting.

```
static void NetworkIPAddr( IPN IPAddr, uint IfIdx, uint fAdd )
{
 ...
    // This is a good time to join any multicast group we require
    if( fAdd && !fAddGroups )
    {
        fAddGroups = 1;
        if (IGMPJoinHostGroup( inet_addr(MulticastAddr), IfIdx ))
            printf("Successfully joined multicast group %s\n", MulticastAddr);
    }
}
...
```

```
int dtask_udp_test (SOCKET s, UINT32 unused)
{
...
    for(;;) {
        // Read the data from port 5000 sent to multicast group
        tmp = sizeof( sin1 );
        sin1.sin_port = htons(5000);
        cnt = (int)recvfrom( s, buffer, 1500, 0, &sin1, &tmp );

        // Check message sent
        if( cnt==15 && !strcmp( (char *)buffer, "Are you there?" ) ) {
            printf("Received message #%d sent by %d.%d.%d.%d\n", ++reply,
                sin1.sin_addr.s_addr & 0xFF, (sin1.sin_addr.s_addr>>8) & 0xFF,
                (sin1.sin_addr.s_addr>>16) & 0xFF,
                (sin1.sin_addr.s_addr>>24) & 0xFF);

            // Reply on port 5001
            sin1.sin_port = htons(5001);
            sendto( s, "I am here!", 11, 0, &sin1, sizeof(sin1) );
        }
    return(1);
}
```

The client “joins” a multicast during the IP network setup. The daemon task (dtask_udp_test)()) listens on port 5000. Once message is received, it replies to the message on a different port (5001).

3.3.1 Building and Running the Client Example

A Code Composer Studio (CCStudio) project file is provided with the example. The project (client-multicast.pjt) is located in the examples/multicast/client/<*board*> folder. The example is provided in source only; you must re-build it for your particular version of the tools supported by the NDK. After building the example, you can simply load and run the example.

You can run multiple client example applications simultaneously. They will all receive the message sent by the simulated “host” and reply accordingly. Ensure that each board running the client example has a unique MAC address.

This example prints (to the standard out window) a string when it receives a multicast message. The message indicates the IP address of the board that sent the multicast message.

```
TCP/IP Multicast Example Client
Using MAC Address: 08-00-28-34-0e-cd
Service Status: DHCPC    : Enabled  :          : 000
Service Status: DHCPC    : Enabled  : Running  : 000
Link Status: 100Mb/s Full Duplex on PHY 1
Network Added: If-1:156.117.95.162
Successfully joined multicast group 224.1.2.3
Service Status: DHCPC    : Enabled  : Running  : 017
Received message #1 sent by 156.117.95.197
Received message #2 sent by 156.117.95.197
```

4 Conclusion

Though the TMS320C6000 Network Developer's Kit (NDK) doesn't support multicasting in a traditional BSD socket way, it is possible to support basic multicasting capabilities as illustrated in the example provided. The NDK supports multicasting from a client perspective only, and is currently limited to 32 multicast addresses, but future NDK releases will address this limit.

5 References

- *TMS320C6000 Network Developer's Kit (NDK) Software User's Guide* (SPRU523)
- *TMS320C6000 Network Developer's Kit (NDK) Software Programmer's Reference Guide* (SPRU524)
- *Unix Network Programming, Vol. 1: The Sockets Networking API, Third Edition* by W. Richard Stevens, Bill Fenner, and Andrew M. Rudoff. Addison-Wesley Professional (2003).
- *Internetworking Technology Handbook*, Chapter 43: Internet Protocol Multicast. http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/ipmulti.pdf, Cisco Systems.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
Low Power Wireless	www.ti.com/lpw	Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265