User's Manual

NetX Duo™

Network Address Translation (NAT)

User Guide

Renesas Synergy™ Platform

Renesas Electronics
www.renesas.com

Nov 8, 2018

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user's manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user's manuals, application notes, "General Notes for Handling and Using Semiconductor Devices" in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

Renesas Synergy Specific Information

If you are using NetX Duo NAT for the Renesas Synergy platform, please use the following information.

Installation

Page 10: If you are using Renesas Synergy SSP and the e^2 studio ISDE, NAT will already be installed. You can ignore the NAT Installation section.

Inbound Entry

Note that inbound entry support for NetX Duo NAT has not been tested for SSP v1.5.0.

Network Address Translation (NAT)

User Guide

Express Logic, Inc.
858.613.6640
Toll Free 888.THREADX
FAX 858.521.4259
www.expresslogic.com

Part Number: 000-1054
Revision 5.11

Contents

Chapter 1 An Introduction to Network Address Translation

The Need for Network Address Translation

IP Network Address Translation (NAT) was originally developed to solve the problem of a limited number of Internet IPv4 addresses. The need for NAT arises when multiple devices need to access the Internet but only one IPv4 Internet address is assigned by the Internet Service Provider (ISP).

There are other benefits of using NAT as well. Network topology outside the local domain can change in many ways. Customers may change providers, company backbones may be reorganized, or providers may merge or split. Whenever the external topology changes, address assignments for hosts within the local domain must also change to reflect these external changes. Changes of this type can be hidden from users within the domain by centralizing changes to a single address translation router. NAT enables access for local hosts to the public Internet and protects them from direct access from the outside. Organizations with a network setup predominantly for internal use, with a need for occasional external access are good candidates for this scheme.

Basic NAT and Network Address Port Translation

A NAT-enabled router is installed between the public network and the private network. The role of the NAT-enabled router is to translate between the internal private IPv4 addresses and the assigned public IPv4 address, so all the devices on the private network are able to share the same public IPv4 address.

In the basic implementation of NAT, the NAT router ‘owns’ one or more globally registered IP addresses different from its own IP address. These global addresses are available to assign to hosts on its private network either statically or dynamically. NAPT, or Network Address Port Translation, is a variation of basic NAT, where network address translation is extended to include a ‘transport’ identifier. Most typically this is the port number for TCP and UDP packets, and the Query ID for ICMP packets.

Connections across the NAT boundary are typically initiated by hosts on the private network sending outbound packets to an external host. These hosts are usually assigned *dynamic* (temporary) IP addresses for this purpose. However, it is also possible to have connections initiated in the opposite direction if the private network has ‘servers’ e.g. HTTP or FTP servers that will accept Client requests from the external network. NAT will typically assign these local hosts a *static* (permanent) IP address:port.

How Network Address Translation Works

A typical network setup with a NAT-enabled router is illustrated in Figure 1.

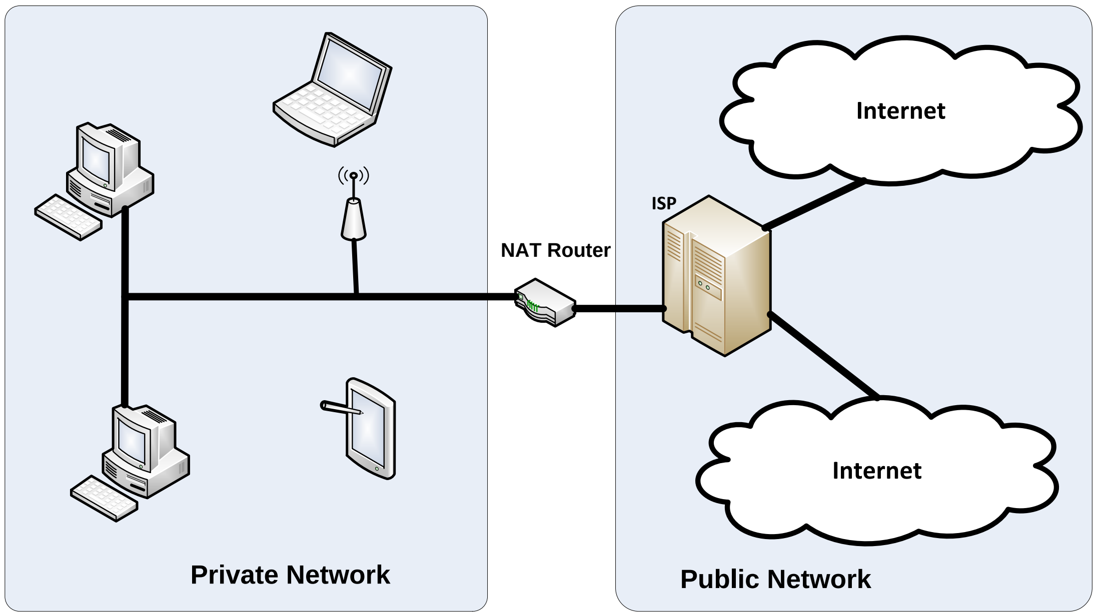

Figure 1

A NAT-enabled router typically has two network interfaces. One interface is connected to the public Internet; the other is connected to the private network. A typical router in this setup is responsible for routing IP datagrams between the private network and the public network based on destination IP address. A NAT-enabled router performs address translation before routing an IPv4 datagram between the public and the private interface. A translation is established for each TCP or UDP session, based the internal source address, source port number, and external destination address and destination port number. For ICMP echo request and response datagram, the ICMP query ID is used instead of the port number.

To illustrate a typical implementation of Network Address Translation, let us consider a network setup in Figure 2.

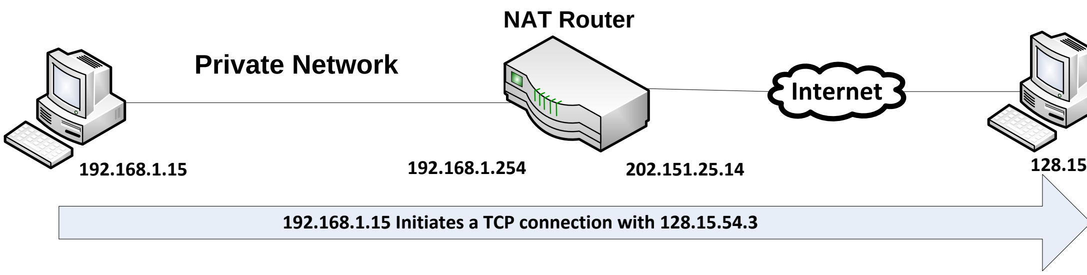

Figure 2

In this scenario, the NAT router connects the private network to the left, and the public network to the right. Let's assume on the public network side, the NAT router interface IP address is 202.151.25.14; on the private network interface, the NAT router uses the IP address 192.168.1.254. A node on the private network initiates a TCP connection with a web server on the Internet.

[might want to spell out non standard abbreviations, SA, SP etc in the diagrams. For non native English readers it might be confusing – janet]

Figure 3 shows a high-level view of the Network Address Translation process.

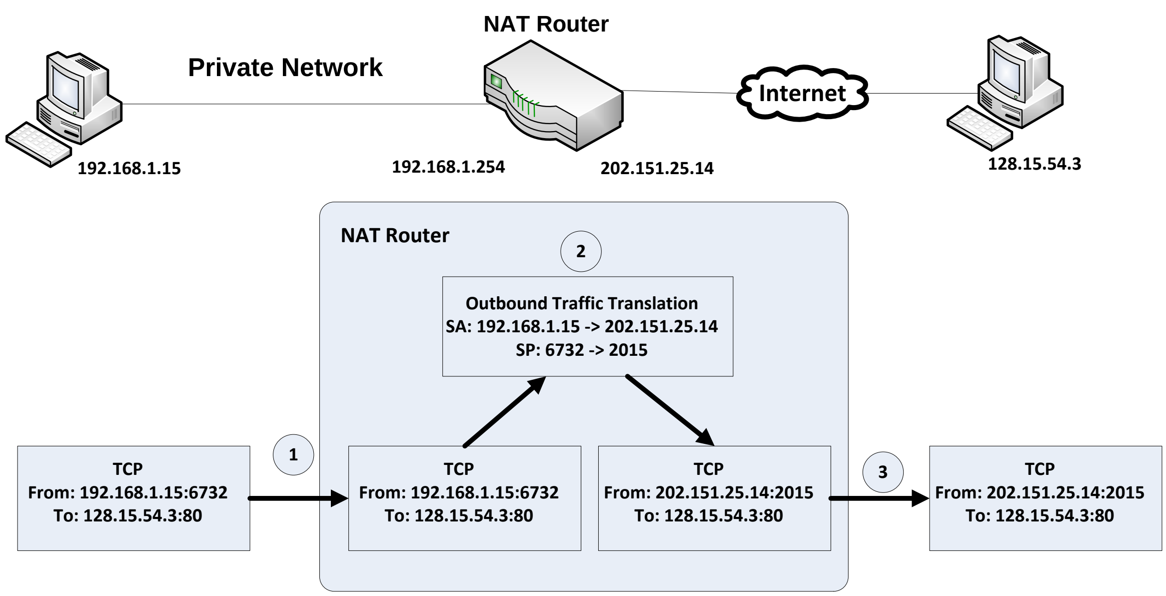

Figure 3

Step 1: Client transmits a TCP SYN message to the web server. The sender address is 192.168.1.15, port number 6732; the destination address is 128.15.54.3, port number 80.

Step 2: The packet from the Client is received on the private network interface by the NAT router. The outbound traffic rule applies to the packet: the sender's (Client's) address is translated to the NAT router's public IP address 202.15.25.14, and sender (Client) source port number is translated to the TCP port number 2015 on the public interface.

Step 3: The packet is then transmitted over the Internet and ultimately reaches its destination host 128.15.54.3. Notice that on the receiving side, based on the IP layer source address and TCP layer port number, the packet appears to have originated from 202.151.24.14, port number 2015.

Figure 4 shows the NAT process on the return path.

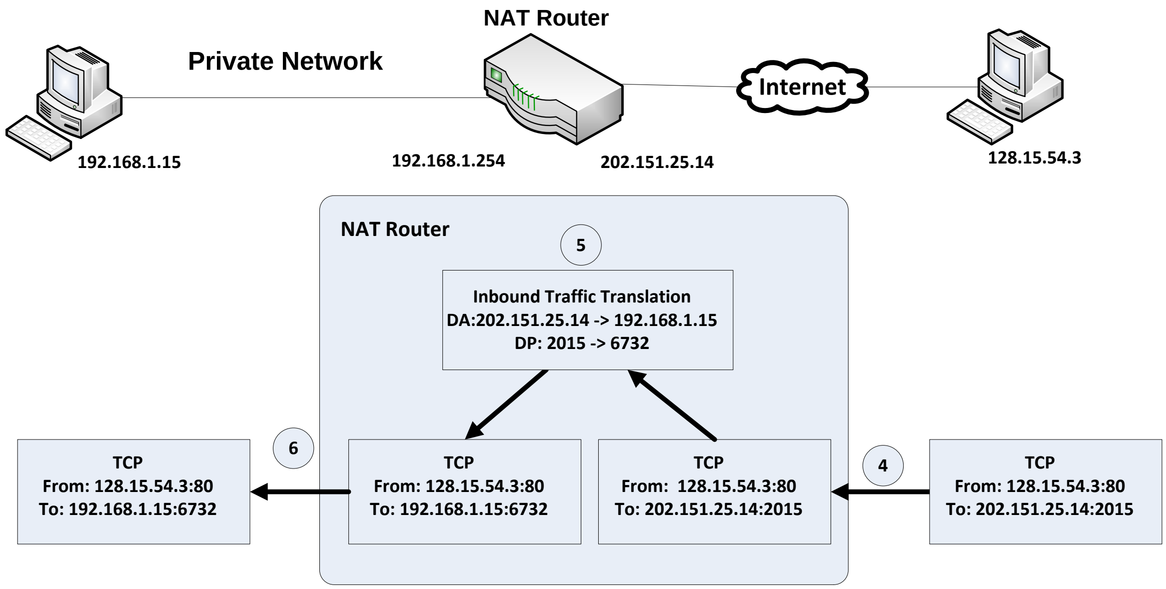

Figure 4

Step 4: In this scenario, the Internet host 128.15.54.3 sends a response packet with the NAT router's Internet address as its destination.

Step 5: The packet reaches the NAT router. Since this is an in-bound packet, the in-bound translation rules apply: the destination address is changed back to the original sender's (Client's) IP address: 192.168.1.15, destination port number 6732.

Step 6: The packet is then forwarded to the Client through the interface that is connected to the internal network.

In this manner the internet network address and port number of the sender is not exposed to other hosts on the public Internet.

NetX Duo NAT Features

When the NAT instance is created using *nx_nat_create* call, the NAT translation table is created.

```
UINT  nx_nat_create(NX_NAT_DEVICE *nat_ptr, NX_IP *ip_ptr,
                    UINT global_interface_index,
                    VOID *dynamic_cache_memory,
                    UINT dynamic_cache_size)
```

To keep track of the network address translations for all active connections between local and external networks, the NetX Duo NAT-enabled router maintains a translation table with information about each private host connection which includes source and destination IP address and port number.

The location of this translation table ("cache") is set with the `dynamic_cache_memory` pointer. This area must be a 4 byte aligned buffer space. The size of the table (or number of entries) is determined by dividing the cache size `dynamic_cache_size` by the size of a NAT table entry. The table must be large enough for the minimal number of entries specified by NX_NAT_MIN_ENTRY_COUNT which is defined in *nx_nat.h*. The default value is 3.

The timeout for all dynamic entries in the NetX Duo NAT translation table are initialized to NX_NAT_ENTRY_RESPONSE_TIMEOUT which is defined in *nx_nat.h*. The default value is 4 minutes (or 240 system ticks for a 100 mHz processor) as recommended by RFC 2663. Each time NetX Duo NAT receives or sends a packet matching a dynamic entry in the table it resets that entry's time out to NX_NAT_ENTRY_RESPONSE_TIMEOUT. When searching the table, NetX Duo NAT will also check the table for expired entries and delete them.

To create inbound entries as static in the table e.g. for servers on the local network, NetX Duo NAT provides the *nx_nat_inbound_entry_create* service. If a table entry defines the local host connection as static, it never expires.

```
UINT  nx_nat_inbound_entry_create(NX_NAT_DEVICE *nat_ptr,
                                  NX_NAT_TRANSLATION_ENTRY *entry_ptr,
                                  ULONG local_ip_address, USHORT external_port,
                                  USHORT local_port, UCHAR protocol)
```

This service is described in more detail in Chapter 4 **Description of Services.**

During runtime, if the translation table is full and no more entries can be added, NetX Duo NAT will notify the NAT application with a cache full callback if one is registered with the NAT instance. This is done using the *nx_nat_cache_notify_set* service:

```
UINT nx_nat_cache_notify_set(NX_NAT_DEVICE *nat_ptr,
              VOID (*cache_full_notify_cb)(NX_NAT_DEVICE *nat_ptr))
```

See Chapter 4 **Description of Services** for more details about this service.

NAT Packet Processing in NetX Duo

NetX Duo NAT is intended for use on an IPv4 router. For NAT to work, NetX Duo must be configured for forwarding packets to the NAT server. See Chapter 2 on NetX Duo NAT installation for how to do so. The NAT server then indicates if it will ‘consume’ (attempt to forward) the packet to a host on either of its networks. If it will not consume the packet, the packet is ‘returned’ to NetX Duo to process the packet as it normally would.

When the NAT server receives a packet to forward from NetX Duo, it determines if the packet is inbound or outbound.

For outbound packets, the NAT server checks the packet IP header source address and port. If the translation table does not contain an entry for a packet previously sent by this host for the same destination, NAT will create a new entry which will contain a unique global source IP address:port for the connection, and modify the packet headers with this new IP address:port before sending it onto the external network.

For inbound packets, the NAT server looks for a previous entry in its translation table with an external IP address: port matching the packet destination IP address: port. If no match is found, it will discard the packet unless the destination address: port is the external address for server on the local network. If it does find a match, it will replace the packet header’s external destination IP address: port with the private IP address: port and send the packet onto the local network to the intended private host.

NetX Duo NAT uses a range of TCP, UDP and ICMP translation ports for creating unique local address: port connections for local hosts connecting with outside hosts. The following user configurable options, defined in *nx_nat.h,* define the range for each protocol:

NX_NAT_START_TCP_PORT
NX_NAT_END_TCP_PORT
NX_NAT_START_UDP_PORT
NX_NAT_END_UDP_PORT
NX_NAT_START_ ICMP_QUERY_ID
NX_NAT_END_ ICMP_QUERY_ID

NAT Requirements and Constraints

NetX Duo NAT requires NetX Duo 5.8 or later. The NAT application requires creation of a single IP instance and an interface to the internal and external physical network.

Constraints:

- NetX Duo NAT supports TCP, UDP and ICMP. IGMP is not supported.
- NetX Duo NAT does not support IPv6 addressing.
- NetX Duo NAT does not include DNS or DHCP services, although NetX Duo NAT can integrate those services with its NAT operations.

RFCs Supported by NetX Duo NAT

NetX Duo NAT implementation is based on information presented in the following RFCs:

- RFC 2663: IP Network Address Translator (NAT) Terminology and Considerations
- RFC 3022: Traditional IP Netowrk Address Translator (Traditional NAT)
- RFC 4787: Network Address Translation (NAT) Behavioral Requirements for Unicast UDP

Chapter 2 Installation and Use of NAT

This chapter contains a description how to install, set up, and use the NetX Duo NAT services.

NetX Duo NAT Installation

NetX Duo NAT is shipped on a single CD-ROM compatible disk. The NetX Duo NAT package includes one source file and one header file, a demonstration application file, and a PDF file for this document, as follows:

nx_nat.c C Source file for NetX Duo NAT
nx_nat.h C Header file for NetX Duo NAT
demo_netx_nat.c Example host NetX Duo C source file
nx_nat.docx Description of the NetX Duo NAT User Guide (this document)

Copy the NetX Duo NAT source code files to the same directory where NetX Duo and ThreadX are installed. For example, if NetX Duo and ThreadX are installed in the directory "*\threadx\mcf5485\green*" then *nx_nat.c*, *nx_nat.h* and the modified NetX Duo files should be copied into this directory. Copy the modified NetX Duo files over the existing NetX Duo files. Copy the Ethernet controller driver files into this directory as well.

To build a NetX Duo NAT application:

- The NetX Duo library *nxduo.a* must be built with NX_NAT_ENABLED defined. This can be done in *nx_user.h*, (make sure NX_INCLUDE_USER_DEFINE_FILE is also defined to ensure that configuration options in *nx_user.h* are included in the build.

- The application project must include *nx_nat.h* after *tx_api.h* and *nx_api.h.* The latter two header files are necessary to use ThreadX and NetX Duo services.

- The application then enables NAT on a previously created IP instance using the *nx_nat_enable* service.

- The application code can dynamically enable/disable NAT by calling the *nx_nat_enable* and *nx_nat_disable* service.

- The application project code is compiled and linked with the NAT enabled NetX Duo library to create the executable.

- To support NAT connections using TCP, UDP or ICMP protocols, NetX Duo must be enabled to support that protocol. This is done by calling *nx_tcp_enable, nx_udp_enable* and *nx_icmp_enable* for the previously created IP instance respectively.

Small Example Demo NAT Setup

An example of how an application sets up NetX Duo NAT is shown in the *tx_application_define* function in Figure 4 below. Unlike most NetX Duo demo files distributed on the installation CD, this demo runs on an actual processor board with two Ethernet controllers, instead of a Windows PC using the virtual network driver *_nx_ram_network_driver*(). The NAT device is connected to the local domain through a local switch on its local interface, and to the external network through second switch on its external interface.

NetXDuo basic configuration is shown in demo_netx_nat.c. The private network is defined as 192.168.2.xx and has two local host nodes. The global network is defined as 192.168.0.xx and defines its gateway for out of network packets as 192.168.0.1. The NetX Duo IP instances are created on lines 118-171 and invoke the 'ram' driver; nat_ip instance attached two interfaces act as an NAT router, local_ip instance attached on interface act as local host; external_ip instance attached one interface act as external host.

The NAT is created in line 252 and invokes the cache to store dynamic translation entries. Enable the NAT feature in line319, static translation entrie (inbound entry) is created in lines 362 to allow external host to access to local host.

```
  1 /*
  2    demo_netx_nat.c
  3
  4    This is a small demo of NAT (Network Address Translation) on the high-performance
  5    NetX TCP/IP stack.  This demo relies on ThreadX, NetX and NAT APIs to perform network
  6    address translation for IP packets traveling between private and external networks.
  7    this demo concentrates on the ICMP ping operation.
  8 */
  9
 10 #include   "tx_api.h"
 11 #include   "nx_api.h"
 12 #include   "nx_nat.h"
 13
 14 extern void    test_control_return(UINT status);
 15 #if defined NX_NAT_ENABLE && defined __PRODUCT_NETXDUO__ &&
(NX_MAX_PHYSICAL_INTERFACES >= 2)
 16
 17 #define     DEMO_STACK_SIZE         2048
 18
 19 /* Define the ThreadX and NetX object control blocks...  */
```

```

static TX_THREAD                    ntest_0;

/* Set up the NAT components. */

/* Create a NAT instance, packet pool and translation table. */

NX_NAT_DEVICE                       nat_server;
NX_IP                               nat_ip;
NX_IP                               local_ip;
NX_IP                               external_ip;
NX_PACKET_POOL                      nat_packet_pool;
UINT                                error_counter = 0;


/* Configure the NAT network parameters. */

/* Set NetX IP packet pool packet size. This should be less than the Maximum Transmit Unit (MTU) of
   the driver (allow enough room for the Ethernet header plus padding bytes for frame alignment).  */
#define NX_NAT_PACKET_SIZE                          1536


/* Set the size of the NAT IP packet pool.  */
#define NX_NAT_PACKET_POOL_SIZE                     (NX_NAT_PACKET_SIZE * 10)

/* Set NetX IP helper thread stack size. */
#define NX_NAT_IP_THREAD_STACK_SIZE                 2048

/* Set the server IP thread priority */
#define NX_NAT_IP_THREAD_PRIORITY                   2

/* Set ARP cache size of a NAT ip instance. */
#define NX_NAT_ARP_CACHE_SIZE                       1024

/* Set NAT entries memory size. */
#define NX_NAT_ENTRY_CACHE_SIZE                     1024

/* Define NAT IP addresses, local host private IP addresses and external host IP address. */
#define NX_NAT_LOCAL_IPADR              (IP_ADDRESS(192, 168, 2, 1))
#define NX_NAT_LOCAL_HOST1              (IP_ADDRESS(192, 168, 2, 3))
#define NX_NAT_LOCAL_HOST2              (IP_ADDRESS(192, 168, 2, 10))
#define NX_NAT_LOCAL_GATEWAY            (IP_ADDRESS(192, 168, 2, 1))
#define NX_NAT_LOCAL_NETMASK            (IP_ADDRESS(255, 255, 255, 0))
#define NX_NAT_EXTERNAL_IPADR           (IP_ADDRESS(192, 168, 0, 10))
#define NX_NAT_EXTERNAL_HOST            (IP_ADDRESS(192, 168, 0, 100))
#define NX_NAT_EXTERNAL_GATEWAY         (IP_ADDRESS(192, 168, 0, 1))
#define NX_NAT_EXTERNAL_NETMASK         (IP_ADDRESS(255, 255, 255, 0))

/* Create NAT structures for creating NAT tables with static
   entries for local server hosts. */
NX_NAT_TRANSLATION_ENTRY            server_inbound_entry_icmp;

/* Define thread prototypes.  */
static void     ntest_0_entry(ULONG thread_input);
extern void    _nx_ram_network_driver(struct NX_IP_DRIVER_STRUCT *driver_req);

/* Define main entry point.  */

int main()
{

    /* Enter the ThreadX kernel.  */
    tx_kernel_enter();
}


/* Define what the initial system looks like.  */

void    tx_application_define(void *first_unused_memory)
{

UINT     status;
UCHAR    *pointer;

    /* Initialize the NetX system. */
    nx_system_initialize();

    /* Setup the pointer to unallocated memory.  */
```

```
    pointer =  (UCHAR *) first_unused_memory;

    /* Create the main thread.  */
    tx_thread_create(&ntest_0, "thread 0", ntest_0_entry, 0,
                     pointer, DEMO_STACK_SIZE,
                     4, 4, TX_NO_TIME_SLICE, TX_AUTO_START);
    pointer =  pointer + DEMO_STACK_SIZE;

    /* Create NAT packet pool. */
    status =  nx_packet_pool_create(&nat_packet_pool, "NAT Packet Pool",
                                    NX_NAT_PACKET_SIZE, pointer,
                                    NX_NAT_PACKET_POOL_SIZE);

    /* Update pointer to unallocated (free) memory. */
    pointer = pointer + NX_NAT_PACKET_POOL_SIZE;

    /* Check status.  */
    if (status)
        return;

    /* Create IP instances for NAT server (global network) */
    status = nx_ip_create(&nat_ip, "NAT IP Instance", NX_NAT_EXTERNAL_IPADR, NX_NAT_EXTERNAL_NETMASK,
                          &nat_packet_pool, _nx_ram_network_driver, pointer,
                          NX_NAT_IP_THREAD_STACK_SIZE, NX_NAT_IP_THREAD_PRIORITY);

    /* Update pointer to unallocated (free) memory. */
    pointer =  pointer + NX_NAT_IP_THREAD_STACK_SIZE;

    /* Check status.  */
    if (status)
    {
        error_counter++;
        return;
    }

    /* Set the private interface(private network).  */
    status += nx_ip_interface_attach(&nat_ip, "Private Interface", NX_NAT_LOCAL_IPADR, NX_NAT_LOCAL_NETMASK, _nx_ram_network_driver);

    /* Check status.  */
    if (status)
    {
        error_counter++;
        return;
    }

    /* Create IP instances for Local network IP instance */
    status = nx_ip_create(&local_ip, "Local IP Instance", NX_NAT_LOCAL_HOST1, NX_NAT_LOCAL_NETMASK,
                          &nat_packet_pool, _nx_ram_network_driver, pointer,
                          NX_NAT_IP_THREAD_STACK_SIZE, NX_NAT_IP_THREAD_PRIORITY);

    /* Update pointer to unallocated (free) memory. */
    pointer =  pointer + NX_NAT_IP_THREAD_STACK_SIZE;

    /* Check status.  */
    if (status)
    {
        error_counter++;
        return;
    }

    /* Create IP instances for external network IP instance */
    status = nx_ip_create(&external_ip, "External IP Instance", NX_NAT_EXTERNAL_HOST, NX_NAT_EXTERNAL_NETMASK,
                          &nat_packet_pool, _nx_ram_network_driver, pointer,
                          NX_NAT_IP_THREAD_STACK_SIZE, NX_NAT_IP_THREAD_PRIORITY);

    /* Update pointer to unallocated (free) memory. */
    pointer =  pointer + NX_NAT_IP_THREAD_STACK_SIZE;

    /* Check status.  */
    if (status)
    {
        error_counter++;
        return;
    }

    /* Set the global network gateway for NAT IP instance.  */
    status = nx_ip_gateway_address_set(&nat_ip, NX_NAT_EXTERNAL_GATEWAY);
```

```c

    /* Check status.  */
    if (status)
    {
        error_counter++;
        return;
    }

    /* Set the global network gateway for Local IP instance.  */
    status = nx_ip_gateway_address_set(&local_ip, NX_NAT_LOCAL_GATEWAY);

    /* Check status.  */
    if (status)
    {
        error_counter++;
        return;
    }

    /* Set the global network gateway for External IP instance.  */
    status = nx_ip_gateway_address_set(&external_ip, NX_NAT_EXTERNAL_GATEWAY);

    /* Check status.  */
    if (status)
    {
        error_counter++;
        return;
    }


    /* Enable ARP and supply ARP cache memory for NAT IP isntance. */
    status =  nx_arp_enable(&nat_ip, (void **) pointer,
                            NX_NAT_ARP_CACHE_SIZE);

    /* Check status.  */
    if (status)
    {
        error_counter++;
        return;
    }

    /* Update pointer to unallocated (free) memory. */
    pointer = pointer + NX_NAT_ARP_CACHE_SIZE;

    /* Enable ARP and supply ARP cache memory for Local IP isntance. */
    status =  nx_arp_enable(&local_ip, (void **) pointer,
                            NX_NAT_ARP_CACHE_SIZE);

    /* Check status.  */
    if (status)
    {
        error_counter++;
        return;
    }

    /* Update pointer to unallocated (free) memory. */
    pointer = pointer + NX_NAT_ARP_CACHE_SIZE;

    /* Enable ARP and supply ARP cache memory for External IP isntance. */
    status =  nx_arp_enable(&external_ip, (void **) pointer,
                            NX_NAT_ARP_CACHE_SIZE);

    /* Check status.  */
    if (status)
    {
        error_counter++;
        return;
    }

    /* Update pointer to unallocated (free) memory. */
    pointer = pointer + NX_NAT_ARP_CACHE_SIZE;

    /* Enable ICMP. */
    nx_icmp_enable(&nat_ip);
    nx_icmp_enable(&local_ip);
    nx_icmp_enable(&external_ip);

    /* Create a NetX NAT server and cache with a global interface index.  */
    status =  nx_nat_create(&nat_server, &nat_ip, 0, pointer, NX_NAT_ENTRY_CACHE_SIZE);

    /* Check status.  */
    if (status)
```

```
    {
        error_counter++;
        return;
    }

    /* Update pointer to unallocated (free) memory. */
    pointer = pointer + NX_NAT_ENTRY_CACHE_SIZE;
}

/* Define the test threads.  */

static void    ntest_0_entry(ULONG thread_input)
{

UINT        status;
NX_PACKET   *my_packet;

    /***********************************/
    /*       Disable NAT feature       */
    /***********************************/
    /* Local Host ping External Host address.  */
    status =  nx_icmp_ping(&local_ip, NX_NAT_EXTERNAL_HOST, "ABCDEFGHIJKLMNOPQRSTUVWXYZ", 28, &my_packet, 100);

    /* Check status.  */
    if (status == NX_SUCCESS)
    {
        error_counter++;
        return;
    }

    /* Check the NAT forwarded count.  */
#ifndef NX_DISABLE_NAT_INFO
    if ((nat_server.forwarded_packets_received != 0) || (nat_server.forwarded_packets_sent != 0) ||(nat_server.forwarded_packets_dropped != 0))
    {
        error_counter++;
        return;
    }
#endif

    /* External Host ping NAT External address, NAT IP instance will response the requet. */
    status = nx_icmp_ping(&external_ip, NX_NAT_EXTERNAL_IPADR, "ABCDEFGHIJKLMNOPQRSTUVWXYZ", 28, &my_packet, 100);

    /* Check status.  */
    if ((status != NX_SUCCESS) || (my_packet == NX_NULL) || (my_packet -> nx_packet_length != 28))
    {
        error_counter++;
        return;

    /* Check the NAT forwarded count.  */
#ifndef NX_DISABLE_NAT_INFO
    if ((nat_server.forwarded_packets_received != 0) || (nat_server.forwarded_packets_sent != 0) ||(nat_server.forwarded_packets_dropped != 0))
    {
        error_counter++;
        return;
    }
#endif
    }

    /***********************************/
    /*       Enable NAT feature        */
    /***********************************/

    /* Enable the NAT service.  */
    nx_nat_enable(&nat_server);

    /* Local Host ping External Host address.  */
    status =  nx_icmp_ping(&local_ip, NX_NAT_EXTERNAL_HOST, "ABCDEFGHIJKLMNOPQRSTUVWXYZ", 28, &my_packet, 100);

    if ((status != NX_SUCCESS) || (my_packet == NX_NULL) || (my_packet -> nx_packet_length != 28))
    {
        error_counter++;
        return;
    }
```

```

    /* Check the NAT forwarded count.  */
#ifndef NX_DISABLE_NAT_INFO
    if ((nat_server.forwarded_packets_received != 2) || (nat_server.forwarded_packets_sent != 2) ||(nat_server.forwarded_packets_dropped != 0))
    {
        error_counter++;
        return;
    }
#endif

    /* External Host ping NAT External address, NAT IP instance will response the requet. */
    status =  nx_icmp_ping(&external_ip, NX_NAT_EXTERNAL_IPADR, "ABCDEFGHIJKLMNOPQRSTUVWXYZ", 28, &my_packet, 100);

    if ((status != NX_SUCCESS) || (my_packet == NX_NULL) || (my_packet -> nx_packet_length != 28))
    {
        error_counter++;
        return;
    }

    /* Check the NAT forwarded count.  NAT receive the ping request, but can not forward this packet to local network.  discard it.  */
#ifndef NX_DISABLE_NAT_INFO
    if ((nat_server.forwarded_packets_received != 3) || (nat_server.forwarded_packets_sent != 2) ||(nat_server.forwarded_packets_dropped != 1))
    {
        error_counter++;
        return;
    }
#endif

    /**********************************************/
    /*       Create an inbound entry for ICMP     */
    /**********************************************/

    /* Calling NAT API to create a inbound entry.  */
    status = nx_nat_inbound_entry_create(&nat_server, &server_inbound_entry_icmp, NX_NAT_LOCAL_HOST1, 0, 0, NX_PROTOCOL_ICMP);

    if (status != NX_SUCCESS)
    {
        error_counter++;
        return;
    }

    /* External Host ping NAT External address, LOCAL HOST1 will response all inbound icmp request from external network.  */
    status =  nx_icmp_ping(&external_ip, NX_NAT_EXTERNAL_IPADR, "ABCDEFGHIJKLMNOPQRSTUVWXYZ", 28, &my_packet, 100);

    if ((status != NX_SUCCESS) || (my_packet == NX_NULL) || (my_packet -> nx_packet_length != 28))
    {
        error_counter++;
        return;
    }

    /* Check the NAT forwarded count.  */
#ifndef NX_DISABLE_NAT_INFO
    if ((nat_server.forwarded_packets_received != 5) || (nat_server.forwarded_packets_sent != 4) ||(nat_server.forwarded_packets_dropped != 1))
    {
        error_counter++;
        return;
    }
#endif
}
#endif
```

Figure 4. Setting up NetX Duo NAT

Chapter 3 NAT Configuration Options

Configurable options for the NetX Duo NAT API can be found in *nx_nat.*h with the exception of the first one, **NX_DISABLE_ERROR_CHECKING** which is found in *nx_nat.c*. The following list includes all options and their function described in detail:

Define	**Meaning**
NX_DISABLE_ERROR_CHECKING	This option if defined removes the basic NAT error checking. It is typically used after the application has been debugged. The default NetX Duo NAT status is defined (enabled).
NX_NAT_ENABLE_REPLACEMENT	This option if defined enables automatic replacement when NAT cache is full. Note: only replace the oldest non-TCP session.
NX_NAT_MIN_ENTRY_COUNT	This option sets the minimum count for translation entry. The default count is 3.
NX_NAT_TCP_SESSION _TIMEOUT	This option sets the timeout for translation entry for TCP Sessions. The default timeout is 24 hours.
NX_NAT_NON_TCP_SESSION_TIMEOUT	This option sets the timeout for translation entry for non-TCP Sessions. The default timeout is 240 seconds.
NX_NAT_START_TCP_PORT	This option sets the starting value for finding an unused TCP port to assign an outbound TCP packet. The default value is 20000.

NX_NAT_END_TCP_PORT

This option sets the upperlimit of TCP port to assign an outbound TCP packet. The default value is 30000.

NX_NAT_START_UDP_PORT

This option sets the starting value for finding an unused UDP port to assign an outbound UDP packet. The default value is 20000.

NX_NAT_END_UDP_PORT

This option sets the upperlimit of UDP port to assign an outbound UDP packet. The default value is 30000.

NX_NAT_START_ICMP_QUERY_ID

This option sets the starting value for finding an unused query ID to assign an outbound ICMP query packet. The default value is 20000.

NX_NAT_END_ICMP_QUERY_ID

This option sets the upperlimit of query IDs to assign an outbound ICMP query packet. The default value is 30000.

Chapter 4 Description of NAT Services

This chapter contains a description of all NetX Duo NAT API services (listed below) in alphabetical order.

In the "Return Values" section in the following API descriptions, values in **BOLD** are not affected by the **NX_DISABLE_ERROR_CHECKING** define that is used to disable API error checking, while non-bold values are completely disabled.

nx_nat_create
Create a NAT Instance

nx_nat_delete
Delete a NAT instance

nx_nat_enable
Enable the NAT server

nx_nat_disable
Disable the NAT server

nx_nat_cache_notify_set
Set function pointers to user defined cache full notify function.

nx_nat_inbound_entry_create
Create a inbound translation table entry

nx_nat_inbound_entry_delete
Delete a inbound translation table entry

nx_nat_create

Create a NAT Server

Prototype

```
UINT    nx_nat_create(NX_NAT_DEVICE *nat_ptr, NX_IP *ip_ptr,
                      UINT global_interface_index,
                      VOID *dynamic_cache_memory,
                      UINT dynamic_cache_size)
```

Description

This service creates an instance of the NAT server.

Input Parameters

nat_ptr	Pointer to NAT instance to create
ip_ptr	Pointer to IP instance
global_interface_index	Index to the global network interface
dynamic_cache_memory	Pointer memory area to NAT table
dynamic_cache_size	Size of memory area for NAT table

Return Values

NX_SUCCESS	(0x00)	NAT server successfully created
NX_PTR_ERROR	(0x07)	Invalid input pointer parameter
NX_NAT_PARAM_ERROR	(0xD01)	Invalid non pointer input
NX_NAT_CACHE_ERROR	(0xD02)	Invalid cache pointer input

Allowed From

Application code

Example

```
#define NX_NAT_ENTRY_CACHE_SIZE 20480

static UCHAR nat_cache[NX_NAT_ENTRY_CACHE_SIZE];
UINT global_interface_index = 0;

/* Create a NAT Server and cache with a global interface index.
*/
status =  nx_nat_create(nat_ptr, ip_ptr, global_interface_index,
                        nat_cache, NX_NAT_ENTRY_CACHE_SIZE);

/* If status = NX_SUCCESS, the NAT instance was successfully
   created. */
```

nx_nat_delete

Delete a NAT Server

Prototype

```
UINT    nx_nat_delete(NX_NAT_DEVICE *nat_ptr)
```

Description

This service deletes a previously created NAT Server.

Input Parameters

nat_ptr	Pointer to NAT instance to delete

Return Values

NX_SUCCESS	(0x00)	NAT successfully deleted
NX_PTR_ERROR	(0x07)	Invalid input pointer parameter

Allowed From

Application code

Example

```
/* Delete the NAT instance. */
status =  nx_nat_delete (nat_ptr);

/* If the NAT instance was successfully deleted, status = NX_SUCCESS. */
```

nx_nat_enable

Enable the IP instance for NAT

Prototype

```
UINT    nx_nat_enable(NX_NAT_DEVICE *nat_ptr)
```

Description

This service enables the IP instance for NAT (e.g. forward received packets to the NAT server).

Input Parameters

nat_ptr	Pointer to NAT instance

Return Values

NX_SUCCESS	(0x00)	NAT successfully enabled
NX_PTR_ERROR	(0x07)	Invalid input pointer parameter

Allowed From

Application code

Example

```
/* Enable the NAT server. */
status =  nx_nat_enable (nat_ptr);

/* If status = NX_SUCCESS, the IP instance was successfully enabled for NAT. */
```

nx_nat_disable

Disable the IP instance for NAT

Prototype

```
UINT    nx_nat_disable(NX_NAT_DEVICE *nat_ptr)
```

Description

This service disables NAT on the IP instance.

Input Parameters

nat_ptr	Pointer to NAT instance

Return Values

NX_SUCCESS	(0x00)	NAT successfully disabled
NX_PTR_ERROR	(0x07)	Invalid input pointer parameter

Allowed From

Application code

Example

```
/* Disable the NAT server. */
status =  nx_nat_disable (nat_ptr);

/* If status = NX_SUCCESS the NAT operation successfully disable. */
```

nx_nat_cache_notify_set

Set a cache full notify callback function

Prototype

```
UINT    nx_nat_cache_notify_set(NX_NAT_DEVICE *nat_ptr,
                        VOID (*cache_full_notify_cb)
                          (NX_NAT_DEVICE *nat_ptr)))
```

Description

This service registers the cache full callback with the input function pointer `cache_full_notify_cb` which points to a user defined cache full notify function.

Input Parameters

nat_ptr	Pointer to NAT instance
cache_full_notify_cb	Pointer to cache full notify function

Return Values

NX_SUCCESS	(0x00)	Cache full notify function successfully set
NX_PTR_ERROR	(0x07)	Invalid input pointer parameter
NX_NAT_PARAM_ERROR	(0xD01)	Invalid non pointer input

Allowed From

Application code

Example

```
/* Set the cache full notify callback function to the NAT instance. */
status = nx_nat_cache_notify_set(nat_ptr, cache_full_notify_cb);

/* If status = NX_SUCCESS the callback function was successfully set. */
```

nx_nat_inbound_entry_create

Create an inbound entry in the NAT translation table

Prototype

```
UINT    nx_nat_inbound_entry_create(NX_NAT_DEVICE *nat_ptr,
                         NX_NAT_TRANSLATION_ENTRY *entry_ptr
                         ULONG local_ip_address,
                         USHORT external_port,
                         USHORT local_port, UCHAR protocol)
```

Description

This service creates an inbound entry as static (permanent entry, never expires) and adds it to the NAT translation table. These entries are usually created for local host servers where a connection is initiated from a host on the outside network. The NAT server checks that the external port is not already in use in the translation table or bound by a previously existing NetX Duo socket of the same protocol.

Input Parameters

nat_ptr	Pointer to NAT instance
entry_ptr	Pointer to translation entry
local_ip_addres	Local host IP address
external_port	Destination port on the external network
local_port	Source (local host) port
protocol	Packet protocol (e.g TCP)

Return Values

NX_SUCCESS	(0x00)	Entry successfully created
NX_NAT_PORT_UNAVAILABLE	(0xD0D)	Invalid external port
NX_PTR_ERROR	(0x07)	Invalid input pointer parameter
NX_NAT_PARAM_ERROR	(0xD01)	Invalid non pointer input

Allowed From

Application code

Example

```
/* Create an entry for an inbound TCP packet. */
status = nx_nat_inbound_entry_create(nat_ptr, entry_ptr,
                              IP_ADDRESS(192,168,2,2), 5001, 5001,
                              NX_PROTOCOL_TCP);
/* If status = NX_SUCCESS the entry was successfully created. */
```

nx_nat_inbound_entry_create

Delete an inbound entry in the NAT translation table

Prototype

```
UINT    nx_nat_entry_delete(NX_NAT_DEVICE *nat_ptr,
                            NX_NAT_TRANSLATION_ENTRY *delete_entry_ptr)
```

Description

This service deletes the specified inbound entry from the translation table.

Input Parameters

nat_ptr	Pointer to NAT instance
delete_entry_ptr	Pointer to the NAT translation entry

Return Values

NX_SUCCESS	(0x00)	Entry successfully deleted
NX_NAT_ENTRY_NOT_FOUND	(0xD04)	Entry does not found
NX_PTR_ERROR	(0x07)	Invalid input pointer parameter
NX_NAT_ENTRY_TYPE_ERROR	(0xD0C)	Invalid translation type

Allowed From

Application code

Example

```
/* Delete the specified static entry from the translation table. */
status = nx_nat_inbound_entry_delete(nat_ptr, delete_entry_ptr);

/* If status = NX_SUCCESS the entry was successfully deleted. */
```

NetX Duo Network Address Translation (NAT) User Guide

Publication Date: Rev.5.12 Nov 8, 2018

Published by: Renesas Electronics Corporation

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics Corporation
TOYOSU FORESIA, 3-2-24 Toyosu, Koto-ku, Tokyo 135-0061, Japan

Renesas Electronics America Inc.
1001 Murphy Ranch Road, Milpitas, CA 95035, U.S.A.
Tel: +1-408-432-8888, Fax: +1-408-434-5351

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709 Quantum Plaza, No.27 ZhichunLu, Haidian District, Beijing, 100191 P. R. China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, 200333 P. R. China
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL 2nd Stage, Indiranagar, Bangalore 560 038, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
17F, KAMCO Yangjae Tower, 262, Gangnam-daero, Gangnam-gu, Seoul, 06265 Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5338

Colophon 5.2

NetX Duo Network Address Translation (NAT)
User Guide

Renesas Electronics Corporation

R11UM0028EU0512