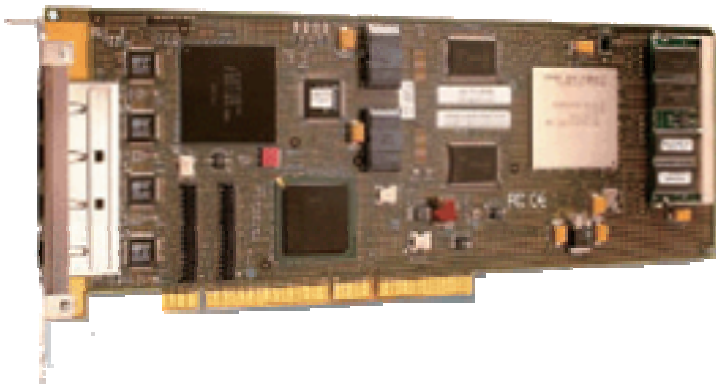
The Dummy's Guide
to IXP1200 Network Processor on ENP-2505



Mel Huang
January 1, 2005
Version 1.0

Copyright Notice

If there is anything wrong or there is any important information not mentioned, any advice is welcomed. Thanks for your direction.
My e-mail: <jasonmel923@yahoo.com.tw>

Table of Contents

Overview

Origin of This Document

This document is designed to provide IXP1200 programmers who have never heard about IXA with a learning direction. It shows what should be known, which document should be read first or not. Hope that this can help the beginners to enter this field more smoothly.

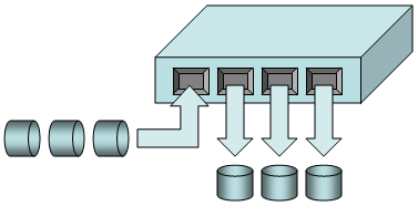
Intended Audience

This document is primarily intended for IXP1200 programmers who have no idea about the whole unknown architecture. It is preferable to have some basic protocol knowledges, router and network processor ideas, and Linux or UNIX operating experences.

Background Knowledge

Router

In packet-switched networks, a router is a device that determines the next network point to which a packet should be forwarded toward its destination, as the figure shown below.



Network Processor

There are 3 ways to implement a router up to now:

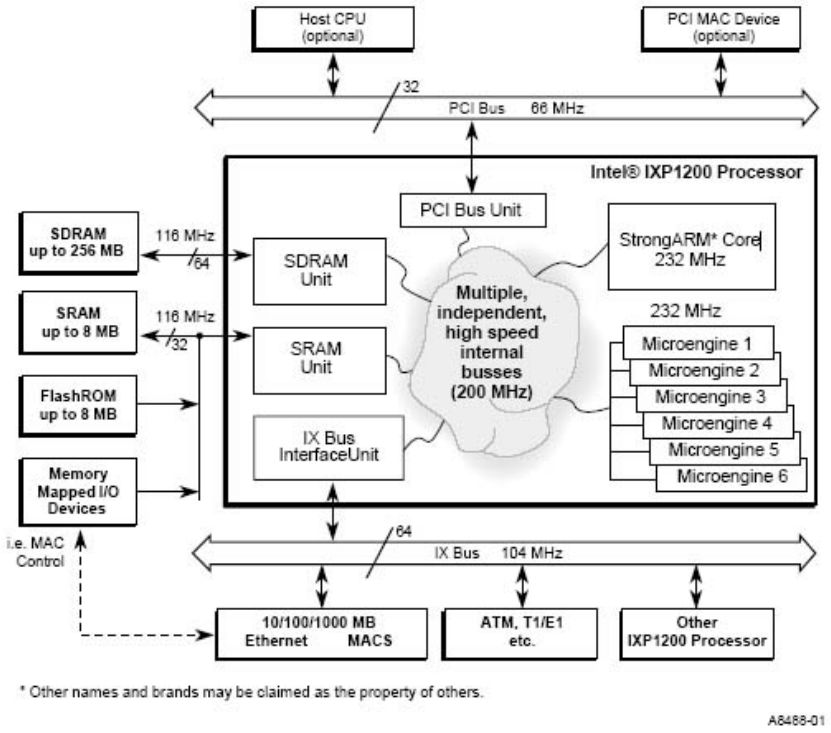
1. Software (Using general PC architecture)
   - :-) Easy to learn ► easy to implement.
   - :-(Instruction set not optimized ► bad performance.

2. ASIC (Application Specific Integrated Circuit)
   - :-) Instruction set optimized ► nice performance.
   - :-(Difficult to learn ► difficult to implement.
   - :-(Long developing period ► high cost.
   - :-(Once burned, never reused ► high cost again.

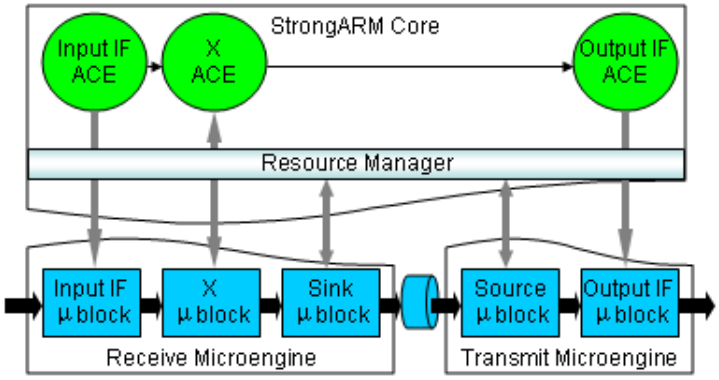
3. Network Processor
   - :-) Instruction set optimized ► nice performance.
   - :-| Difficult to learn ► but easy to implement.
   - :-) Short developing period ► low cost.
   - :-) Can be reused until broken ► low cost again.
     (However, it is a little bit expensive to buy one NP.)

According to the above-mentioned, Network Processor is the best solution to implement a router, especially an experimenting and researching router.

IXP1200 Hardware Architecture



Software Architecture

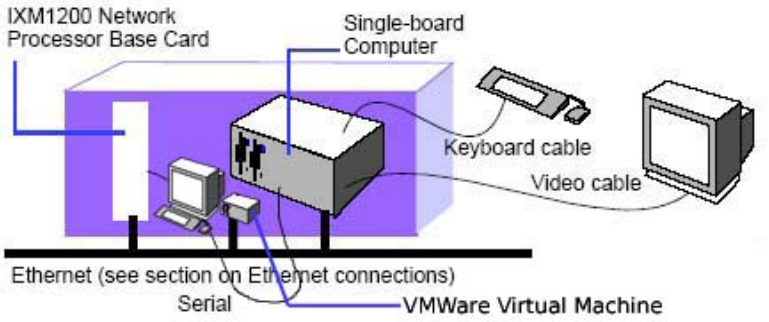


For More Information

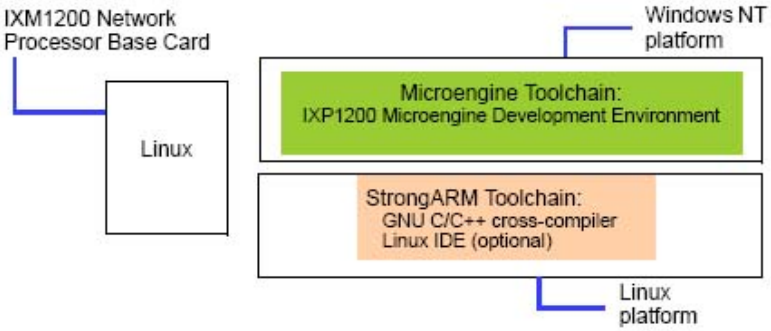
1. Hardware Reference Manual (IXP1200 HW Ref Manual.pdf).
2. SDK 2.01 Tutorial (SDK_Tutorial.pdf).

Environment

Hardware Configuration



Software Configuration



For More Information

1. IXA SDK 2.01 Getting Started (GettingStarted.pdf).

Installation and Setup

Configuring ROM based components

1. Once done, always done.

Configuring the Linux Host

1. Install RedHat 7.3
   - Boot Loader: GRUB
   - Package Selection: NFS, Software Development, Kernel Development, Windows Compatibility/Interoperability. (Check detail for tftp, dhcp.)

2. Upgrading the Host Linux Kernel
   - Login with root.

   - Update kernel:
     $ rpm -recompile linux_24_17src.rpm
     (Located at ENP Linux SDK CD/ENP_LINUX_SDK/HOST_LINUX_UPGRADE/LINUX_24_17SRC.RPM)

   - Modify /etc/grub.conf

```
default=1
title RedHat Linux (2.4.7)
        root (hd0, 0)
        kernel vmlinuz-2.4.7-10 ro root=/dev/hda0
        initrd /initrd-2.4.7-10.img
title RedHat Linux (2.4.17)
        root (hd0, 0)
        kernel vmlinuz-2.4.17-10 ro root=/dev/hda0
        initrd /initrd-2.4.7-10.img
```

   - Set tftp service (/etc/xinetd.d/tftp)

```
service tftp
{
        socket_type             = dgram
        protocol                = udp
        wait                    = wait
        user                    = nobody
        log_on_success          += USERID
        log_on_failure          += USERID
        server          = /usr/sbin/in.tftpd
        server_args             = /tftpboot -l
        disabled                = no
}
```

- Default kernel didn't support non-intel network cards and iptables. Hence, we need to rebuild the kernel.
  $ cd /usr/src/linux-2.4.17
  $ make menuconfig (or make xconfig)
    Network devices ► Realtek RTL8139 or others...
    Networking options ► Network packet filtering (replaces ipchains)
    Networking options ► IP: Netfilter Configuration ► IP tables support...
    Networking options ► IP: Netfilter Configuration ► Connection tracking (required for masq/NAT)
    Networking options ► IP: Netfilter Configuration ► Full NAT ► MASQUERADE
  $ make dep
  $ make clean
  $ make bzImage
  $ make modules
  $ make modules_install
  $ make install
  (Modify /etc/grub.conf again if needed.)

- Reboot.

Install the Intel IXA SDK

1. Installing the IXA SDK on the Linux Host
   - Login with root.

   - Mount IXA SDK VOLII CD1.
     $ rpm -ivh ixa.sdk-2.01.D-fcs.i386.rpm
     $ rpm -ivh ixa.executive-2.01.D-fcs.i386.rpm
     $ rpm -ivh armbe-v4b-fcs.i386.rpm
     $ rpm -ivh ixa.binaries-2.01.D-fcs.i386.rpm
     $ rpm -ivh nfs-utils-0.3.1-13.i386.rpm

   - Modify /etc/profile and add following lines.
     # IXA SDK path settings
     if ! echo $PATH | /bin/grep -q "/usr/local/armbe/bin" ; then
         PATH="$PATH:/usr/local/armbe/bin"
     fi
     if ! echo $PATH | /bin/grep -q "/opt/ixasdk/bin" ; then
         PATH="$PATH:/opt/ixasdk/bin"
     fi
     IXROOT="/opt/ixasdk"
     CONFIG="ARM_BE"
     export IXROOT CONFIG

2. Installing IXA SDK on the Windows Development Workstation
   - ENP Linux SDK CD ► Next ► Next ► ...
     Install portmap.
     Do not enter license key for 30 days trial.

   - Cygwin choose install everything.

3. VMware
   - VMware communicate with Linux by NAT.

   - Set virtual network card vmnet8 as 192.168.x.x. (vmware-config.pl)

   - Set iptables to make Workbench able to communicate with IXP1200 embedded Linux.
     Modify /etc/sysctl.conf ► variable net.ipv4.ip_forward = 1
     $ echo 1 > /proc/sys/net/ipv4/ip_forward
     $ /sbin/iptables -t nat -A POSTROUTING -o vmnet8 -s 192.168.0.2 -j MASQUERADE

ENP Configuration

1. ENP Software Installation
   - $ rpm -recompile boot_drv_201_3src.rpm
     (Located at ENP Linux SDK CD/ENP_LINUX_SDK/ENP_2505_DRIVER/BOOT_DRV201_3SRC.RPM)

   - This step will install ENP-2505/6 drivers, ENP's embedded Linux(2.3.99), and boot scripts to /opt/ixasdk/enp-2505.

2. Rebuilding the ENP Linux kernel
   - $ cd /opt/ixasdk/enp2505/src/linuxIXAedu

   - $ make menuconfig
     - Select "System and processor types"
     - Under "IXP Board type" select ENP-2505
     - Select "Save and exit"

   - $ make dep; make clean; make zImage

   - This step will copy the zImage kernel into /tftpboot for the PCI download scripts.

3. Applying the IXA SDK patch
   - $ cd /opt

   - $ tar -zcvf ixasdk.tgz ixasdk

- $ cp enp_changes_ixa_sdk.path /opt
  (Located at ENP_UPDATES_FOR_IXA_SDK/.)

- cd /opt/ixasdk

- cat ../enp_changes_ixa_sdk.patch | patch -pl

4. Rebuilding the pciDg dirver module
   - Modify /opt/ixasdk/enp-2505/src/pciDg/rules.mk
     ARM-TOOLS = /usr/local/armbe/bin/armv4b-unknown-linux-
     ARM-KERNEL-INCLUDES = $(PWD)/../**linuxIXAedu**/include
     X86-KERNEL-INCLUDES =
     (Or ln -s /opt/ixasdk/enp-2505/src/**l**inuxIXAedu /opt/ixasdk/enp-2505/src/**l**inux)

   - $ make clean; make

5. Rebuilding the SDK
   - Modify /opt/ixasdk/src/arch.incl ► IX_TARGET_TYPE=ENP2505.

   - Rebuilding the MicroAce framework.
     $ cd /opt/ixasdk/src/microace
     $ make clean; make

ENP2505/6 Specific Configuration

1. Installing the ENP-2505/6 IXP1200 PCI Board

2. Booting the ENP-2505/6
   - $ cd /opt/ixasdk/enp-2505/bootixp

   - $ ./bootixp

(It's done up to now if everything is right. Steps below can be ignored.)
3. Verifying ENP-2505/6 Driver operation
   - After prompting “** Finished ENP-2505 driver install **”
     $ lsmod

     On ENP:

Module	Size	Used by
pciDgNet-arm	2668	1
pciDg-arm	7544	0 [pciDgNet-arm]

     On Linux:

Module	Size	Used by
pciDgNet1920	1	
pciDg	5568	0 [pciDgNet]

4. ping 192.168.0.4(Linux) 192.168.0.2(ENP)
   - IP address can be changed at /opt/src/bootxixp/create_environment.rc.

For More Information

1. IXA SDK 2.01 Installation and Setup Guide (SDK_Install.pdf).

Running Sample Applications

Running Simple Count Application

1. Compiling the Microcode Source Files (Under Windows)
   - Using Makefile.win (need to do at first time)
     - Open Cygwin.
     - Check the environment variables:
       $IXROOT = /opt/ixasdk
       $CONFIG = ARM_BE
     - $ cd $IXROOT/src/microace/aces/tutorial1/ucbuild
     - $ make -f Makefile.win
     - ftp to linux:
       ftp> cd /opt/ixasdk/bin/arm-be
       ftp> mput *.uof

   - Using IXP1200 Microengine Development Environment (debugging phase)
     - Open Workbench.
     - Open project
       $IXROOT\src\microace\projects\Count_8_1\Count_8_1.dwp.
     - Build.
     - Debug -> Hardware
       Hardware -> Option -> Connect via Ethernet
       (Enter ENP's IP: 192.168.0.2)

2. Compiling the Core Component of the MicroACE (Under Linux)
   - Check the environment variables:
     $IXROOT = /opt/ixasdk
     $CONFIG = ARM_BE
     $IXPSDKROOT = C:/ixp1200 (not needed?)
   - $ cd $IXROOT/src/microace/aces/tutorial1/count_ace1
   - $ (make clean;) make
   - $ cd $IXROOT/src/microace/projects/Count_8_1
   - Edit ixsys_count_8_1.config if needed.
   - $ (make clean;) make
     (This will export ixsys_count_8_1.config to $IXROOT/bin/arm_be.)

3. Starting and Running the Core Application (Under embedded Linux)
   - $ cd /nfs/ixasdk/bin/arm-be
   - $ ./ixstart ixsys_count_8_1.config
     (If needed, Start Debugging at workbench side.)
   - $ ./ixstop

Running L3 Forwarder Application

1. Compiling the Microcode Source Files (Under Windows)
   - Using Makefile.win (need to do at first time)
     - Open Cygwin.
     - Check the environment variables:
       $IXROOT = /opt/ixasdk
       $CONFIG = ARM_BE
     - $ cd $IXROOT/src/microace/aces/tutorial1/ucbuild
     - $ make -f Makefile.win
     - ftp to linux:
       ftp> cd /opt/ixasdk/bin/arm-be
       ftp> mput *.uof

2. Compiling the Core Component of the MicroACE (Under Linux)
   - Check the environment variables:
     $IXROOT = /opt/ixasdk
     $CONFIG = ARM_BE
     $IXPSDKROOT = C:/ixp1200 (not needed?)
   - $ cd $IXROOT/src/microace/apps/l3config
   - $ (make clean;) make
   - $ $IXROOT/src/microace/aces/l3forward_ace
   - Edit ixsys.config-l3fwdr if needed.
   - $ (make clean;) make
     (This will export ixsys.config-l3fwdr to $IXROOT/bin/arm_be.)

3. Starting and Running the Core Application (Under embedded Linux)
   - $ cd /nfs/ixasdk/bin/arm-be
   - $ ./ixstart ixsys.config-l3fwdr
     (If needed, Start Debugging at workbench side.)
   - $ ./ixstop

For More Information

1. SDK 2.01 Tutorial (SDK_Tutorial.pdf).

Configuration File

```
# ************************************************************************
#     It is run at boot time and specifies
#     - 1. the [interfaces] that are to be started at system boot time
#     - 2. the [micro aces] that are to be started at system boot time
#     - 3. the [regular aces] that are to be started at system boot time
#     - 4. [bind] configuration
#     - 5. [Shell] commands to be run
#
# ************************************************************************
# 1. First we specify the [interfaces] we want
#
# For the SI board    - 0-15 are fast ethernet ports. 16 and 17 are gigabit ports.
# <port number> <ip addr> <broadcast> <netmask> <mac address> <flags>
#
# Values for flags are
#       0x0       Unicast
#       0x1    Promiscous Mode
#       0x2    All Multicast packets are allowed
#       0x3       Multicast packets in set only
#
# ************************************************************************
#       ENP-3511    Port 0-7       10/100
#       ENP-2505    Port 0-3       10/100 (*)
#       ENP-2506    Port 0-1       1000

interface 0 10.1.0.1 10.1.0.255 255.255.255.0 00:01:02:03:04:05 1
interface 1 10.2.0.1 10.2.0.255 255.255.255.0 00:01:02:03:04:06 1
interface 2 10.3.0.1 10.3.0.255 255.255.255.0 00:01:02:03:04:07 1
interface 3 10.4.0.1 10.4.0.255 255.255.255.0 00:01:02:03:04:08 1
#interface 4 10.5.0.1 10.5.0.255 255.255.255.0 00:01:02:03:04:09 1
#interface 5 10.6.0.1 10.6.0.255 255.255.255.0 00:01:02:03:04:10 1
#interface 6 10.7.0.1 10.7.0.255 255.255.255.0 00:01:02:03:04:11 1
#interface 7 10.8.0.1 10.8.0.255 255.255.255.0 00:01:02:03:04:12 1
#interface 8 10.9.0.1 10.9.0.255 255.255.255.0 00:01:02:03:04:13 1
#interface 9 10.10.0.1 10.10.0.255 255.255.255.0 00:01:02:03:04:14 1
#interface 10 10.11.0.1 10.11.0.255 255.255.255.0 00:01:02:03:04:15 1
#interface 11 10.12.0.1 10.12.0.255 255.255.255.0 00:01:02:03:04:16 1
#interface 12 10.13.0.1 10.13.0.255 255.255.255.0 00:01:02:03:04:17 1
#interface 13 10.14.0.1 10.14.0.255 255.255.255.0 00:01:02:03:04:18 1
#interface 14 10.15.0.1 10.15.0.255 255.255.255.0 00:01:02:03:04:19 1
#interface 15 10.16.0.1 10.16.0.255 255.255.255.0 00:01:02:03:04:20 1
#interface 16 10.17.0.1 10.17.0.255 255.255.255.0 00:01:02:03:04:21 1
#interface 17 10.18.0.1 10.18.0.255 255.255.255.0 00:01:02:03:04:22 1
```

```
# ************************************************************************
# Specify if we will be debugging with the workbench and downloading
# code via it
#
# mode <mode>
#
# Values for mode are
#       0x0    No workbench.
#       0x1    Download and debug via workbench

#mode 0
mode 1

# ************************************************************************
# Next specify the Microcode files (UOF Files)
#
# <fileType> <fileName>
#
# Values for fileType are
#       0x0    Slow Ingress File
#       0x1    Slow Egress File
#       0x2    Fast Ingress File for Port 1
#       0x3    Fast Ingress File for Port 2
#       0x4    Fast Egress File

file 0 ./SlowIngressCount.uof
file 1 ./SlowEgressRR.uof
file 2 ./FastIngressCount-seq1.uof
file 3 ./FastIngressCount-seq2.uof
file 4 ./FastEgressFifo.uof

# ************************************************************************
# 2. Now specify the microaces
#
# <name of ace> <name of executable> <config file name> <runsOn> \
#       <type> <additional parameters>
#
# runsOn can be -- 0 (RUNS_ON_INGRESS_SIDE)
#                  -- 1 (RUNS_ON_EGRESS_SIDE)
#
# type can be -- 0 (unknown type)
#                1 (Ingress)
#                2 (egress)
#                3 (L3)
#                4 (L2)
```

```
#                   5 (input nat)
#                   6 (output nat)
#
# The first parameter to every microace is the mask of microengines it
# runs on. This is passed automatically by the application. The <additional
# parameters> specify the rest of the command line parameters

# no config file for ingress and egress needed and no parameters

microace ifaceInput ./ingressAce none 0 1
microace ifaceOutput ./egressAce none 1 2

# This microace will substitute for L3 for now
#
# options for CountMicroAce are
#
# -v : verbose
# -target : send packets over the default target
# -loop : loop back packets to the microblock
# -blockname <name> : name of the microblock
# -port <portNumber>: output port to send to (default is dont change what's in
#                                 the ofnum of the packet)
#

microace CountMicroAce ./CountMicroAce 0 0 none -target -blockname COUNT

# ************************************************************************
# 3. Now specify the regular aces like stack ace, nat control ace, spanning
# tree ace etc
#
# <name of ace> <name of executable> <config file name> <type> <params>
#
# type can be -- 0 (unknown type)
#                   1 (Spanning Tree)
#                   2 (Nat control)
#                   3 (Stack ace)
#
# <params> are any command line (argc/argv) parameters to be passed to
# the ace
# ************************************************************************

# ************************************************************************
# 4. Now specify the all the bind operations
#
# <static/regular> <target name> <ace name>
#
```

```
# Specify static for microace to microace static targets

bind static ifaceInput/default CountMicroAce
bind static CountMicroAce/default ifaceOutput

# ************************************************************************
# 5. Now specify any shell commands. These will always be executed last
# no matter where they are specified. Note that these are to be used
# typically for things like adding arp entries to the linux stack.
#
# Below we add arp entries to the linux stack. This is needed because
# the traffic generator we are using does not respond to ARP requests.
# Otherwise this is typically not needed as long as the ARP target of
# of the interface ace is bound to the stack ace
#
# Below we add a couple of arp entries
#
# Syntax is
#
# sh [command string to be executed]

# sh arp -s 10.1.0.2 01:02:03:04:05:06
# sh arp -s 10.2.0.2 02:02:03:04:05:06
# sh arp -s 10.3.0.2 03:02:03:04:05:06
# sh arp -s 10.4.0.2 04:02:03:04:05:06
# sh arp -s 10.5.0.2 05:02:03:04:05:06
# sh arp -s 10.6.0.2 06:02:03:04:05:06
# sh arp -s 10.7.0.2 07:02:03:04:05:06
# sh arp -s 10.8.0.2 08:02:03:04:05:06
# sh arp -s 10.9.0.2 09:02:03:04:05:06
# ************************************************************************
```

For More Information

1. Configuring and Starting IXA Systems (Chapter 4 of SDK_Guide.pdf).

FAQ

1. Why it always halts on “login:” when booting embedded linux?
Ans: Check if there is something wrong about the serial cable.
(Use a healthy one to try if it works.)

2. What should I do if the embedded linux window is closed uncarefully?
Ans: Reboot is the best solution.

3. Why can't I get correct SRAM variable value in debugging mode?
Ans: It seems a normal condition. Try to copy it to another variable.

GNU Free Document License

In order to limit the size of this document, a copy of the GNU Free Documentation Version 1.1 has not been included within the document itself. A copy of the license is freely available for download from the Free Software Foundation:

http://www.gnu.org/copyleft/fdl.html

Alternatively, a copy of the license terms can be obtained by writing the Free Software Foundation at:

Free Software Foundation, Inc.
59 Temple Place, Suite 330,
Boston, MA 02111-1307
USA

References

MicroAssembly
Microcode Programmer's Reference Manual (IXP1200 Prog Ref Manual.pdf)

MicroC
Microengine C Compiler Language Reference Manual (C_Lang_Ref.pdf)

ENP-2505 Hardware Reference
http://www.radisys.com/files/support_downloads/007-01266-0002.ENP-2505.pdf