基于 OMAPL138 的多核软件开发组件 MCSDK 开发入门

Revision History

Draft Date	Revision No.	Description
2016/05/25	V1.1	1.模板更新。
2013/12/25	V1.0	1.初始版本。

目 录

1 MCSDK 介绍

德州仪器（TI）2013 年 11 月推出基于低功耗 OMAP-L138 DSP+ARM9™处理器的多核软件开发组件——MCSDK（Multicore Software Development Kits），帮助开发人员缩短开发时间，实现针对 TI TMS320C6000™高性能数字信号处理器(DSP)的扩展。为工业、通信、电信以及医疗市场开发各种应用的客户现在无需转移其它软件平台，便可升级至高性能器件。

TI MCSDK 提供高度优化的特定平台基础驱动器捆绑包，可实现基于 TI 器件的开发。此外，MCSDK 还可为实现便捷编程提供定义明确的应用编程接口，支持未来向更高性能的 TI 多核平台的移植，因此开发人员无需从头设计通用层。MCSDK 不仅可帮助开发人员评估特定器件开发平台的软硬件功能，而且还可帮助他们快速开发多核应用。此外，它还有助于应用在统一平台上使用 SYS/BIOS 或 Linux。MCSDK 的各内核通常还可指定运行 Linux 应用，作为控制平台，而其它内核则可同时分配高性能信号处理工作。借助这种异构配置的高灵活性，软件开发人员可在 TI 多核处理器上实施全面解决方案。在 TI OMAP-L138 应用实例中，内部 ARM9 处理器可分配嵌入式 Linux 等高级操作系统执行复杂的 IO 协议栈处理，而 TMS320C674x DSP 则可运行 TI RTOS（上述 SYS/BIOS）实时处理任务。

TI DSP 业务经理 Ramesh Kumar 指出：“能为 OMAP-L138 处理器提供 MCSDK 我们深感振奋。新老客户都将受益，包括在整个 TI C6000™ DSP 中可使用相同的软件、支持编程高效率、加速产品上市进程以及更高的投资回报等。”

MCSDK 包含的库兼容于 TI C674x DSP 以及基于 KeyStone™的 DSP，其中包括 C665x、C667x、66AK2Hx 以及 66AK2Ex 处理器。有了 MCSDK，开发人员可获得各种优化型 DSP 库，包括数学库、数字信号处理库、影像视频处理库、电信库以及语音视频编解码器等，并可从中获益。此外，TI OMAP-L138 处理器还具有应用优化型特性与外设的独特组合，包括以太网、USB、SATA、视频端口接口(VPIF)以及 uPP 等。

2 MCSDK 软件安装

光盘中提供的内核源代码共有三个版本：linux-3.3、linux-2.6.37 和 linux-2.6.33，此文档同时提供了三个源码的编译方法。各版本内核支持特性和维护信息见光盘"Linux/linux-feture-support.xls"文件。三个版本内核的基本区别如下：

linux-3.3：对应 MCSDK 双核开发包，使用 SYSLINK 组件，DSP 端使用 SYS/BIOS。

linux-2.6.37：对应 DVSDK 双核开发包，使用 DSPLINK 组件，DSP 端使用 DSP/BIOS。

linux-2.6.33：没有对应的软件包，使用 DSPLINK 组件，DSP 端使用 DSP/BIOS。

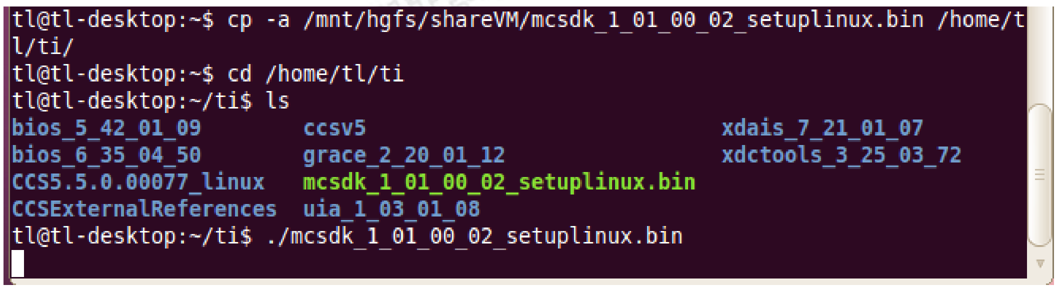
因为 SYS/BIOS 属于 DSP/BIOS 的升级版本，MCSDK 也是最新的多核软件开发包，所以推荐使用 linux-3.3 内核和 MCSDK/SYSLINK 进行开发，其他版本不再推荐使用。广州创龙现阶段主要维护基于 linux-3.3 内核和 MCSDK/SYSLINK 开发的相关代码。

在进行如下步骤前，先按照相关软件安装文件安装 Linux 版本 CCS 和搭建好 tftp、nfs 开发环境，然后在光盘资料 tools 目录中找到"mcsdk_1_01_00_02_setuplinux.bin"安装文件，先将其复制到共享目录，然后执行如下命令：

Host# cp -a /mnt/hgfs/shareVM/mcsdk_1_01_00_02_setuplinux.bin /home/tl/ti/

Host# cd /home/tl/ti

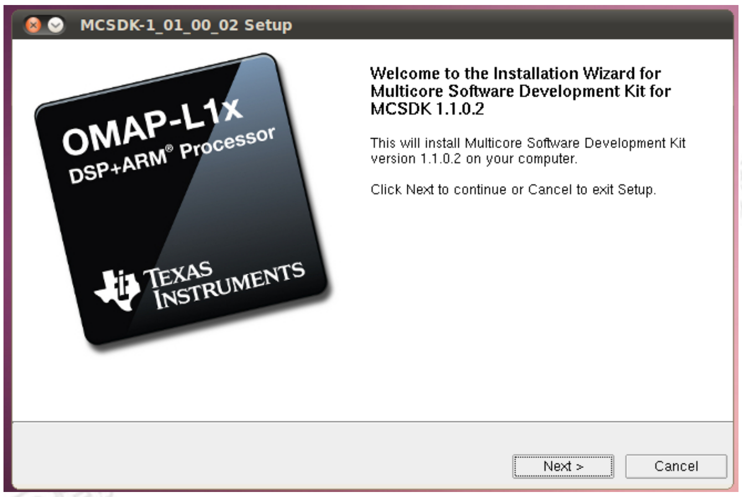
Host# ./mcsdk_1_01_00_02_setuplinux.bin



```
tl@tl-desktop:~$ cp -a /mnt/hgfs/shareVM/mcsdk_1_01_00_02_setuplinux.bin /home/t
l/ti/
tl@tl-desktop:~$ cd /home/tl/ti
tl@tl-desktop:~/ti$ ls
bios_5_42_01_09         ccsv5                            xdais_7_21_01_07
bios_6_35_04_50         grace_2_20_01_12                 xdctools_3_25_03_72
CCS5.5.0.00077_linux    mcsdk_1_01_00_02_setuplinux.bin
CCSExternalReferences   uia_1_03_01_08
tl@tl-desktop:~/ti$ ./mcsdk_1_01_00_02_setuplinux.bin
```

图 1

弹出如下界面，点击 Next。



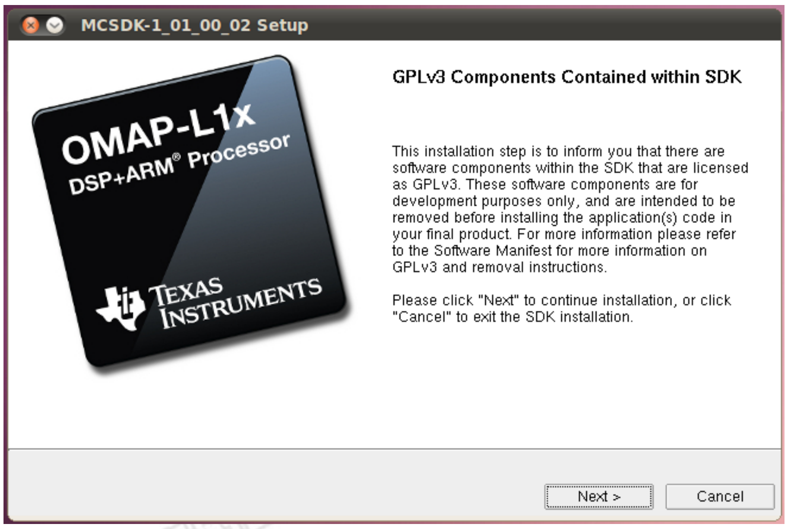
Welcome to the Installation Wizard for Multicore Software Development Kit for MCSDK 1.1.0.2

This will install Multicore Software Development Kit version 1.1.0.2 on your computer.

Click Next to continue or Cancel to exit Setup.

图 2

弹出如下界面，点击 Next。



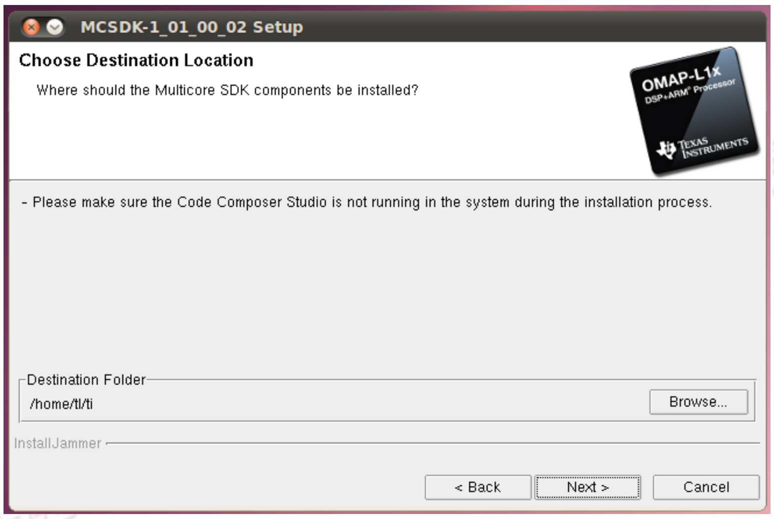
GPLv3 Components Contained within SDK

This installation step is to inform you that there are software components within the SDK that are licensed as GPLv3. These software components are for development purposes only, and are intended to be removed before installing the application(s) code in your final product. For more information please refer to the Software Manifest for more information on GPLv3 and removal instructions.

Please click "Next" to continue installation, or click "Cancel" to exit the SDK installation.

图 3

安装路经选择默认，即"/home/tl/ti"，确保此路径为 CCS 的安装路径，如不同，请修改为一致，然后点击 Next。


MCSDK-1_01_00_02 Setup

Choose Destination Location

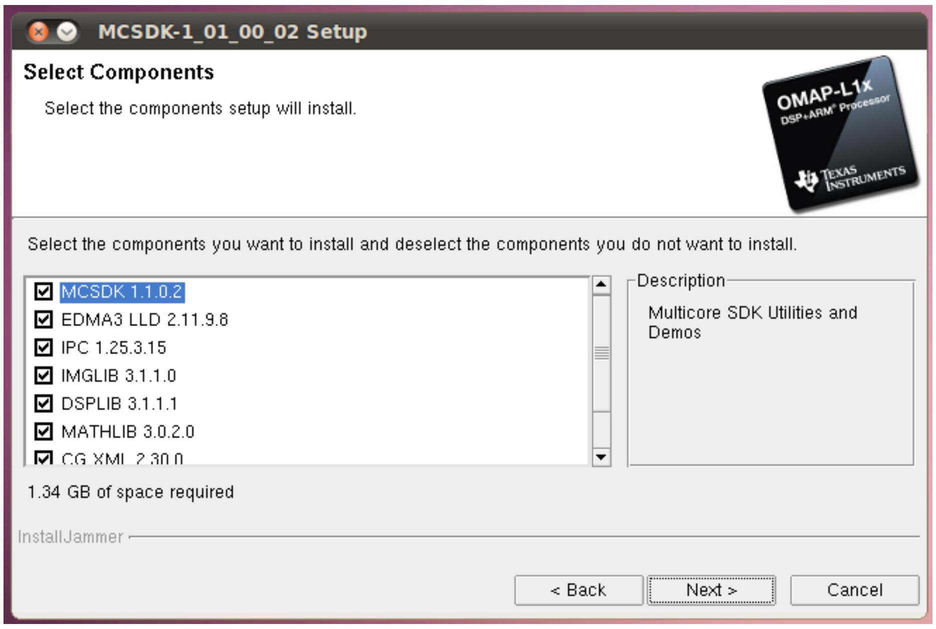
Where should the Multicore SDK components be installed?

- Please make sure the Code Composer Studio is not running in the system during the installation process.

Destination Folder
/home/tl/ti

图 4

弹出如下界面，点击 Next。


MCSDK-1_01_00_02 Setup

Select Components

Select the components setup will install.

Select the components you want to install and deselect the components you do not want to install.

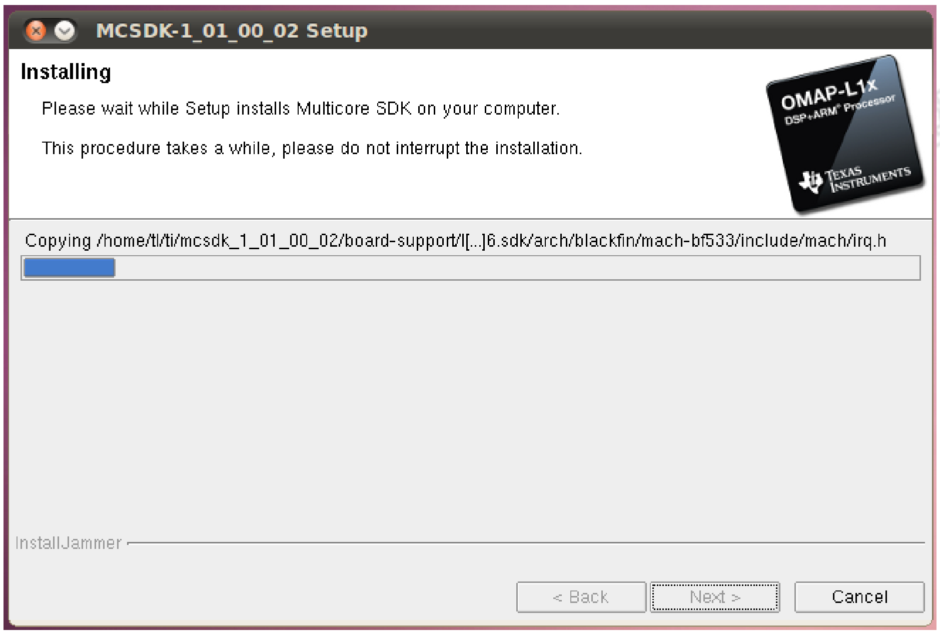
- ☑ MCSDK 1.1.0.2
- ☑ EDMA3 LLD 2.11.9.8
- ☑ IPC 1.25.3.15
- ☑ IMGLIB 3.1.1.0
- ☑ DSPLIB 3.1.1.1
- ☑ MATHLIB 3.0.2.0
- ☑ CG XML 2.30.0

Description
Multicore SDK Utilities and Demos

1.34 GB of space required

图 5

弹出如下界面，开始安装 MCSDK。


MCSDK-1_01_00_02 Setup

Installing

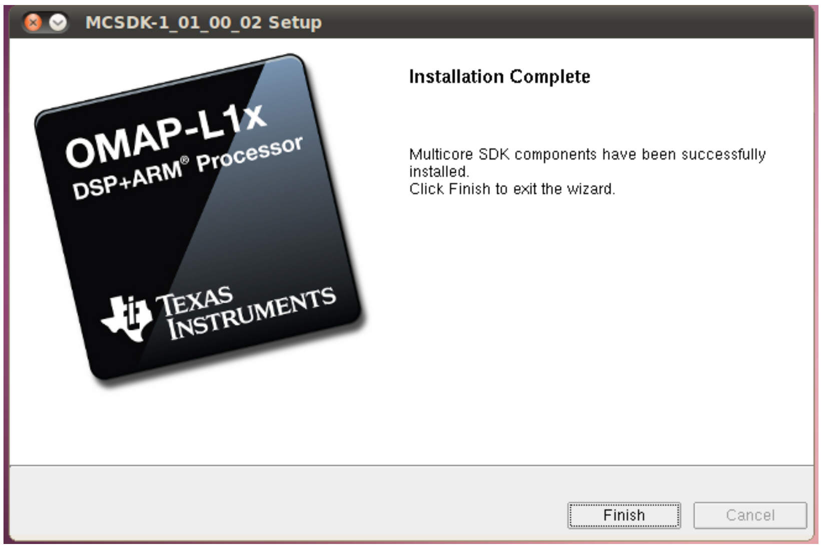
Please wait while Setup installs Multicore SDK on your computer.

This procedure takes a while, please do not interrupt the installation.

Copying /home/tl/ti/mcsdk_1_01_00_02/board-support/l[...]6.sdk/arch/blackfin/mach-bf533/include/mach/irq.h

图 6

等待如下界面出现，点击 Finish 完成安装。


MCSDK-1_01_00_02 Setup

Installation Complete

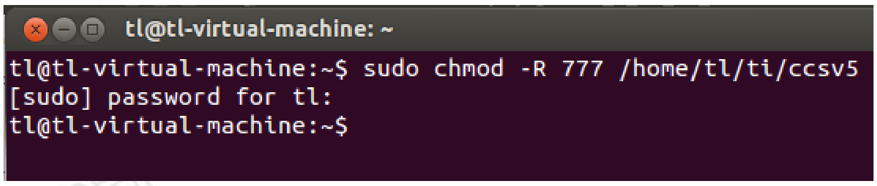
Multicore SDK components have been successfully installed.
Click Finish to exit the wizard.

图 7

3 设置 MCSDK 开发环境参数

执行相关指令，设置 CCS 权限，如图所示：

Host# sudo chmod -R 777 /home/tl/ti/ccsv5



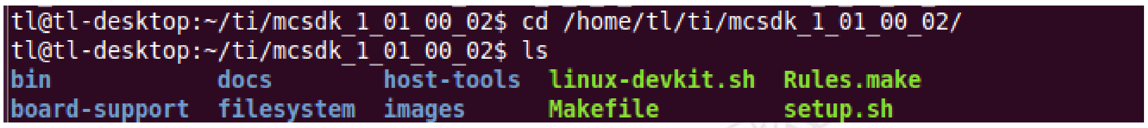
```
tl@tl-virtual-machine:~$ sudo chmod -R 777 /home/tl/ti/ccsv5
[sudo] password for tl:
tl@tl-virtual-machine:~$
```

图 8

进入"mcsdk_1_01_00_02"目录下，启动 MCSDK 设置脚本，根据不同主机设置，进行tftp、nfs、U-Boot 等配置。在设置之前，务必保证虚拟机网络畅通。

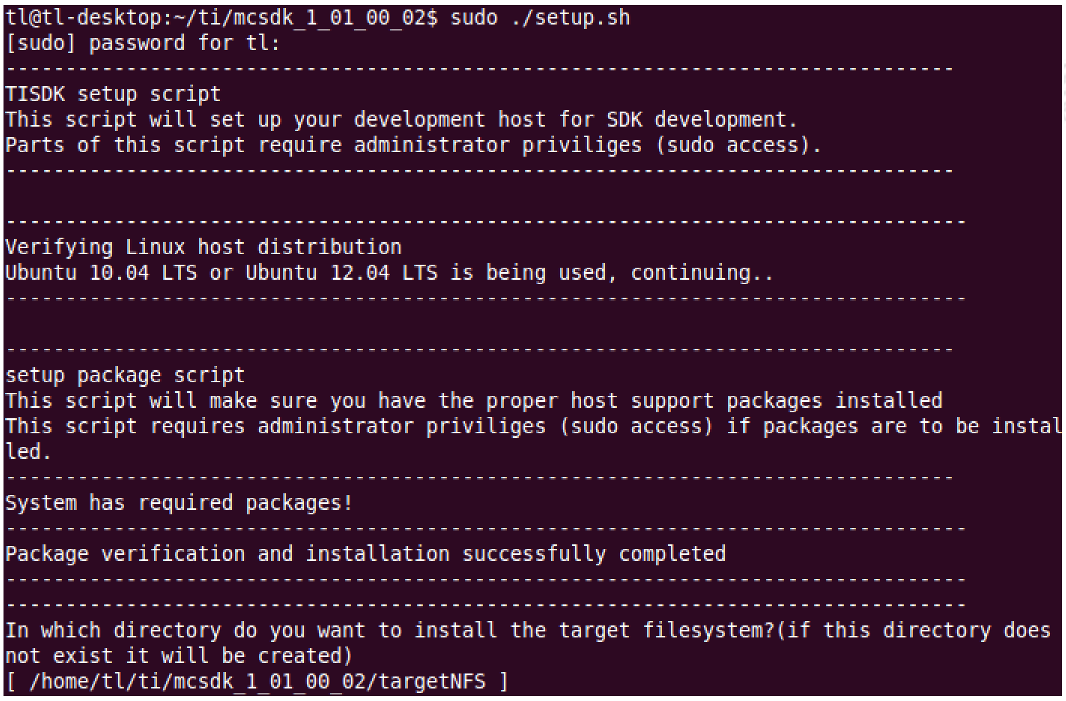
Host# cd /home/tl/ti/mcsdk_1_01_00_02/

Host# sudo ./setup.sh



```
tl@tl-desktop:~/ti/mcsdk_1_01_00_02$ cd /home/tl/ti/mcsdk_1_01_00_02/
tl@tl-desktop:~/ti/mcsdk_1_01_00_02$ ls
bin            docs        host-tools   linux-devkit.sh  Rules.make
board-support  filesystem  images       Makefile         setup.sh
```

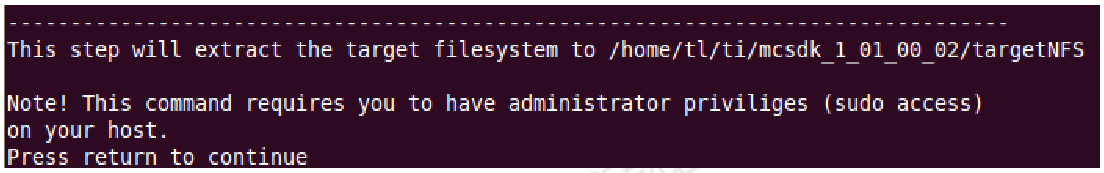
图 9



```
tl@tl-desktop:~/ti/mcsdk_1_01_00_02$ sudo ./setup.sh
[sudo] password for tl:
--------------------------------------------------------------------------------
TISDK setup script
This script will set up your development host for SDK development.
Parts of this script require administrator priviliges (sudo access).
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
Verifying Linux host distribution
Ubuntu 10.04 LTS or Ubuntu 12.04 LTS is being used, continuing..
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
setup package script
This script will make sure you have the proper host support packages installed
This script requires administrator priviliges (sudo access) if packages are to be instal
led.
--------------------------------------------------------------------------------
System has required packages!
--------------------------------------------------------------------------------
Package verification and installation successfully completed
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
In which directory do you want to install the target filesystem?(if this directory does
not exist it will be created)
[ /home/tl/ti/mcsdk_1_01_00_02/targetNFS ]
```

图 10

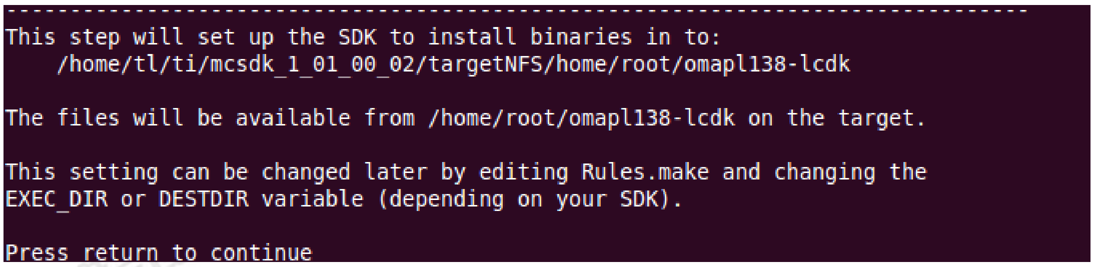
按 Enter 键将 MCSDK 的文件系统安装到默认路径，出现如下界面：



```
--------------------------------------------------------------------------------
This step will extract the target filesystem to /home/tl/ti/mcsdk_1_01_00_02/targetNFS

Note! This command requires you to have administrator priviliges (sudo access)
on your host.
Press return to continue
```

图 11

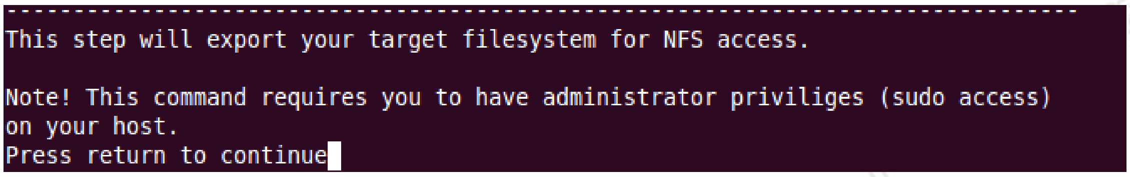
按 Enter 键，出现如下界面：



```
--------------------------------------------------------------------------------
This step will set up the SDK to install binaries in to:
    /home/tl/ti/mcsdk_1_01_00_02/targetNFS/home/root/omapl138-lcdk

The files will be available from /home/root/omapl138-lcdk on the target.

This setting can be changed later by editing Rules.make and changing the
EXEC_DIR or DESTDIR variable (depending on your SDK).

Press return to continue
```

图 12

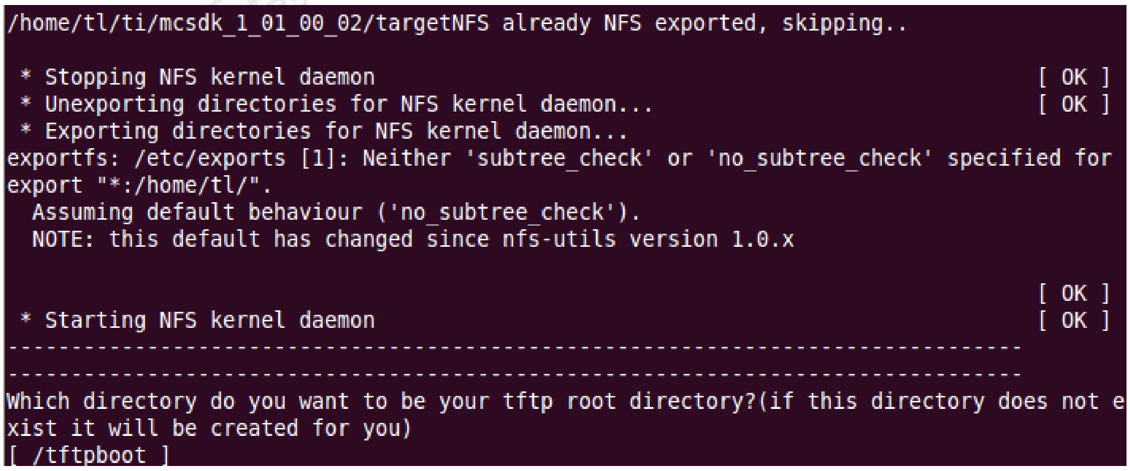
按 Enter 键将 MCSDK 的 Linux 内核镜像安装到默认路径，出现如下界面：



```
--------------------------------------------------------------------------------
This step will export your target filesystem for NFS access.

Note! This command requires you to have administrator priviliges (sudo access)
on your host.
Press return to continue
```

图 13

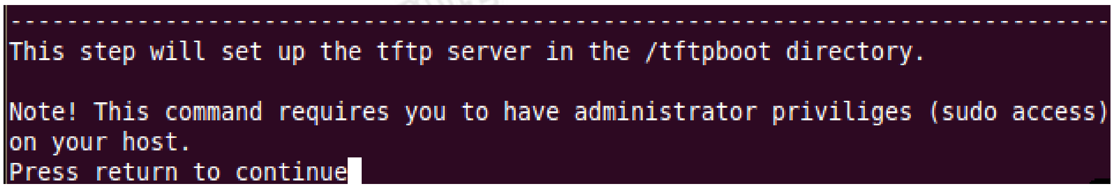
按 Enter 键设置可以进行 nfs 访问，出现如下界面：



```
/home/tl/ti/mcsdk_1_01_00_02/targetNFS already NFS exported, skipping..

 * Stopping NFS kernel daemon                                           [ OK ]
 * Unexporting directories for NFS kernel daemon...                     [ OK ]
 * Exporting directories for NFS kernel daemon...
exportfs: /etc/exports [1]: Neither 'subtree_check' or 'no_subtree_check' specified for
export "*:/home/tl/".
  Assuming default behaviour ('no_subtree_check').
  NOTE: this default has changed since nfs-utils version 1.0.x

                                                                        [ OK ]
 * Starting NFS kernel daemon                                           [ OK ]
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Which directory do you want to be your tftp root directory?(if this directory does not e
xist it will be created for you)
[ /tftpboot ]
```

图 14

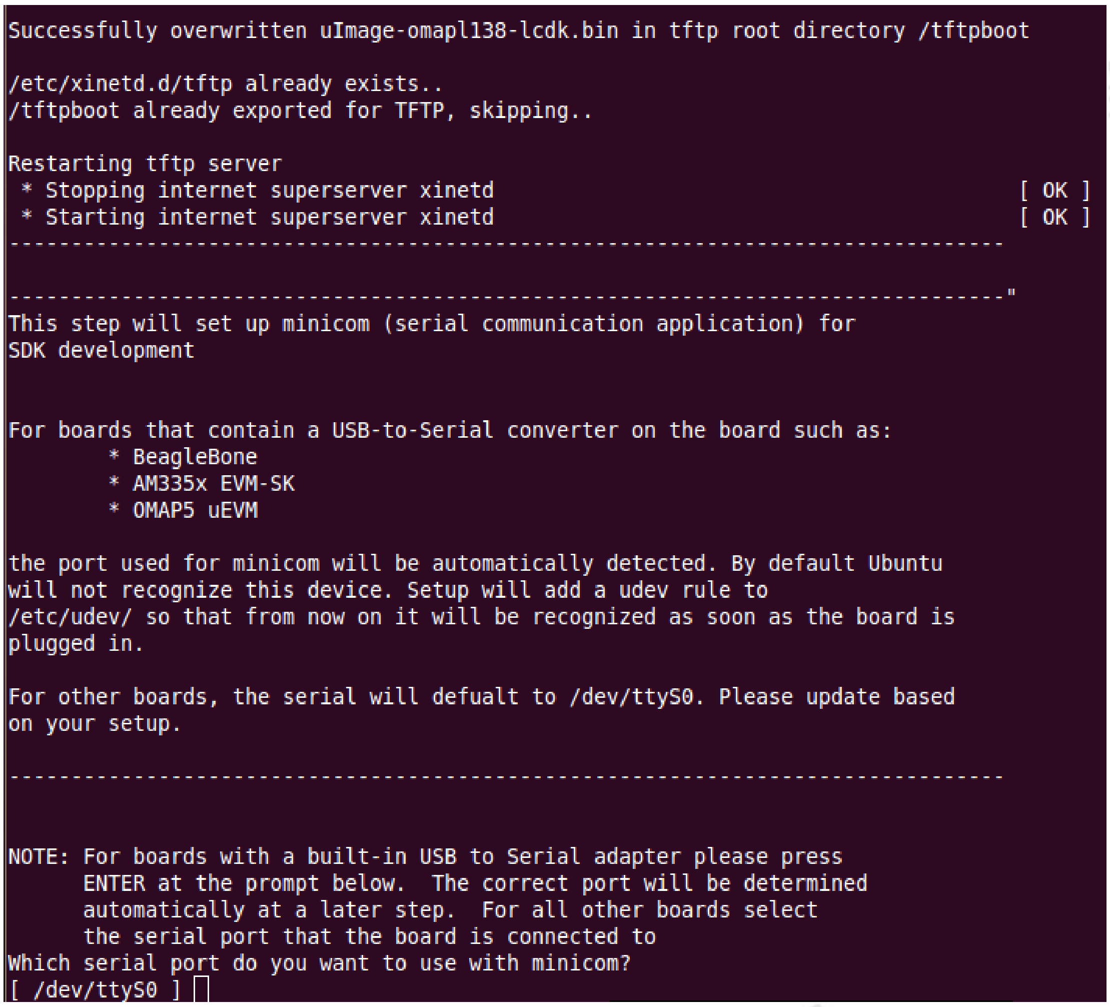
按 Enter 键设置 tftp 服务器下载目录为默认路径（/tftpboot），出现如下界面：



```
--------------------------------------------------------------------------------
This step will set up the tftp server in the /tftpboot directory.

Note! This command requires you to have administrator priviliges (sudo access)
on your host.
Press return to continue
```

图 15

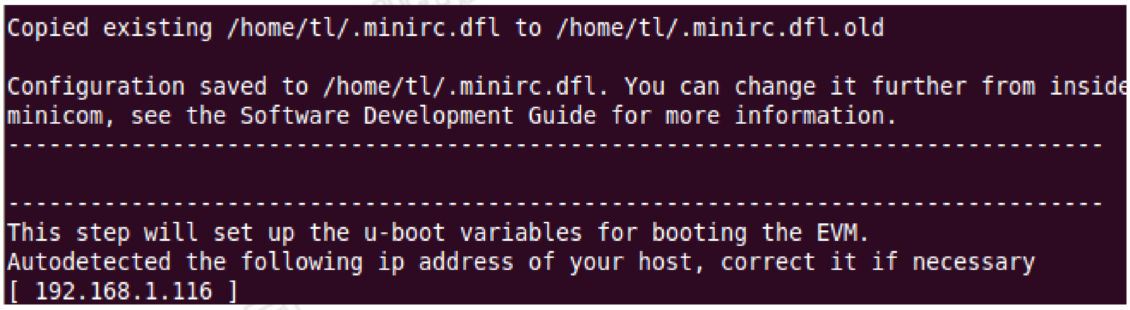
按 Enter 键，出现如下界面：



```
Successfully overwritten uImage-omapl138-lcdk.bin in tftp root directory /tftpboot

/etc/xinetd.d/tftp already exists..
/tftpboot already exported for TFTP, skipping..

Restarting tftp server
 * Stopping internet superserver xinetd                                      [ OK ]
 * Starting internet superserver xinetd                                      [ OK ]
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------"
This step will set up minicom (serial communication application) for
SDK development


For boards that contain a USB-to-Serial converter on the board such as:
        * BeagleBone
        * AM335x EVM-SK
        * OMAP5 uEVM

the port used for minicom will be automatically detected. By default Ubuntu
will not recognize this device. Setup will add a udev rule to
/etc/udev/ so that from now on it will be recognized as soon as the board is
plugged in.

For other boards, the serial will defualt to /dev/ttyS0. Please update based
on your setup.

--------------------------------------------------------------------------------


NOTE: For boards with a built-in USB to Serial adapter please press
      ENTER at the prompt below.  The correct port will be determined
      automatically at a later step.  For all other boards select
      the serial port that the board is connected to
Which serial port do you want to use with minicom?
[ /dev/ttyS0 ]
```

图 16

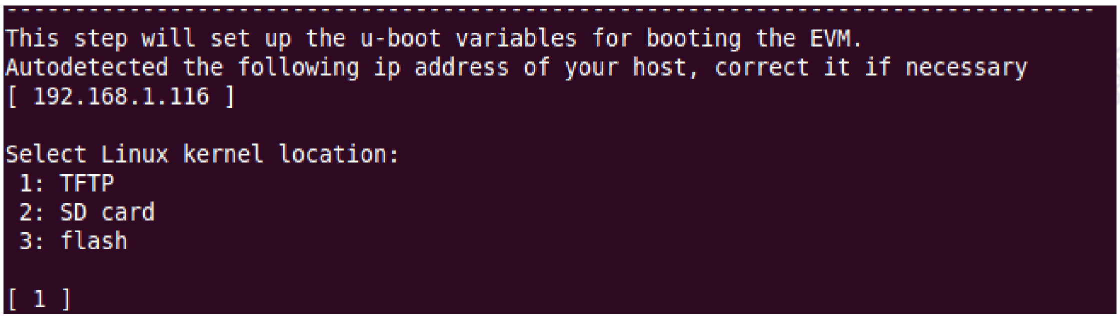
按 Enter 键设置串口为默认设置，出现如下界面：



```
Copied existing /home/tl/.minirc.dfl to /home/tl/.minirc.dfl.old

Configuration saved to /home/tl/.minirc.dfl. You can change it further from inside
minicom, see the Software Development Guide for more information.
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
This step will set up the u-boot variables for booting the EVM.
Autodetected the following ip address of your host, correct it if necessary
[ 192.168.1.116 ]
```

图 17

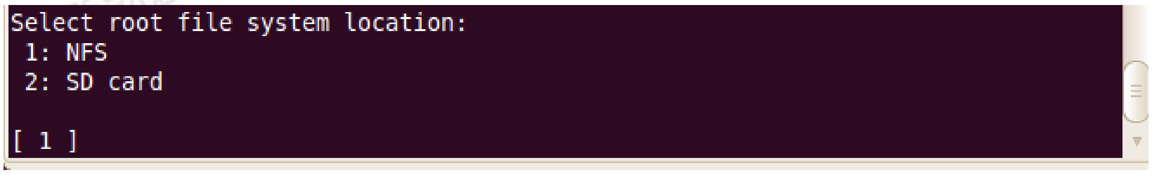
按 Enter 键设置默认的 U-Boot、tftp 网络变量，出现如下界面：



```
--------------------------------------------------------------------------------
This step will set up the u-boot variables for booting the EVM.
Autodetected the following ip address of your host, correct it if necessary
[ 192.168.1.116 ]

Select Linux kernel location:
 1: TFTP
 2: SD card
 3: flash

[ 1 ]
```

图 18

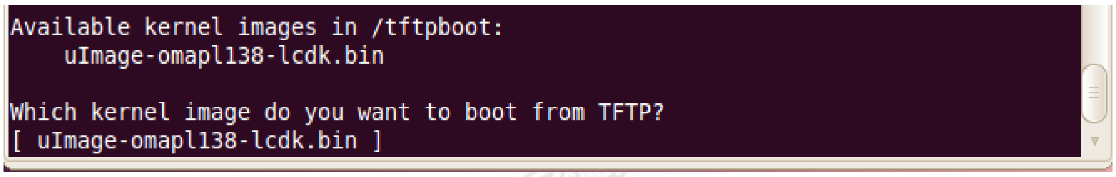
按 Enter 键变量中设置为默认的 nfs 启动方式，出现如下界面：



```
Select root file system location:
 1: NFS
 2: SD card

[ 1 ]
```

图 19

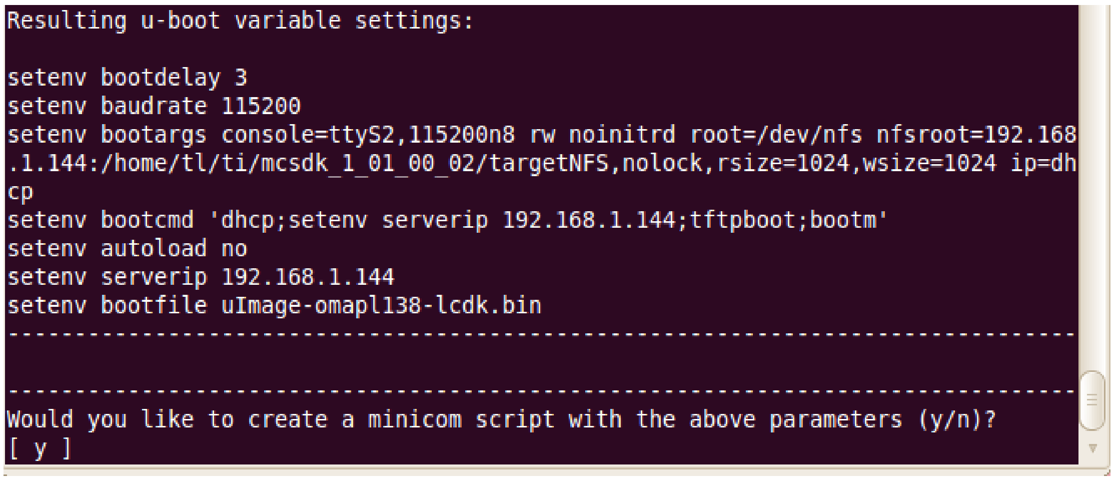
按 Enter 键设置为默认的 nfs 文件系统启动方式，出现如下界面：



```
Available kernel images in /tftpboot:
    uImage-omapl138-lcdk.bin

Which kernel image do you want to boot from TFTP?
[ uImage-omapl138-lcdk.bin ]
```

图 20

按 Enter 键设置启动时 tftp 下载的为默认内核镜像，出现如下界面：



```
Resulting u-boot variable settings:

setenv bootdelay 3
setenv baudrate 115200
setenv bootargs console=ttyS2,115200n8 rw noinitrd root=/dev/nfs nfsroot=192.168
.1.144:/home/tl/ti/mcsdk_1_01_00_02/targetNFS,nolock,rsize=1024,wsize=1024 ip=dh
cp
setenv bootcmd 'dhcp;setenv serverip 192.168.1.144;tftpboot;bootm'
setenv autoload no
setenv serverip 192.168.1.144
setenv bootfile uImage-omapl138-lcdk.bin
--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
Would you like to create a minicom script with the above parameters (y/n)?
[ y ]
```

图 21

输入：n，然后按 Enter 键，出现如下界面：

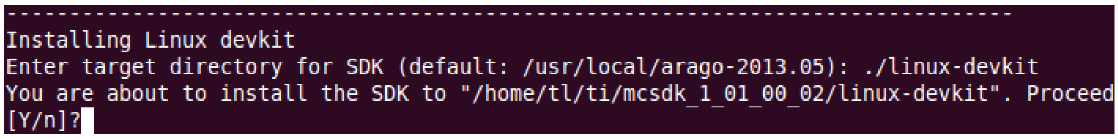

```
--------------------------------------------------------------------------------
Installing Linux devkit
Enter target directory for SDK (default: /usr/local/arago-2013.05): ./linux-devkit
You are about to install the SDK to "/home/tl/ti/mcsdk_1_01_00_02/linux-devkit". Proceed
[Y/n]?
```

图 22

输入：Y，然后按 Enter 键， 现如下界面：

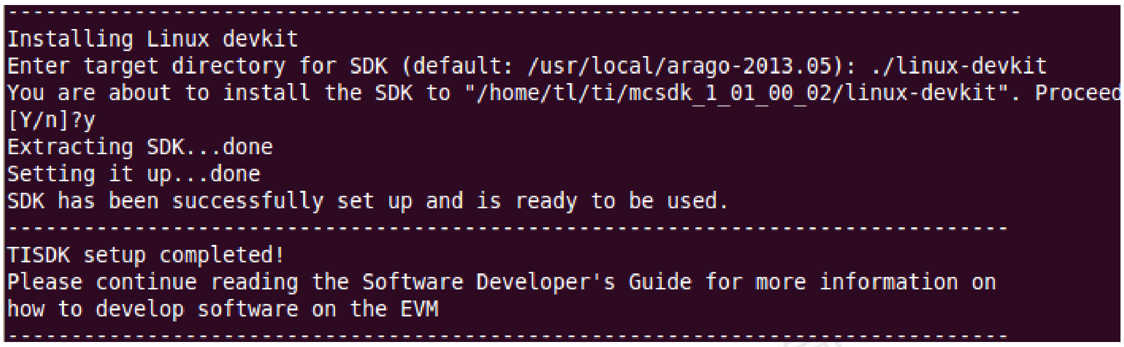

```
--------------------------------------------------------------------------------
Installing Linux devkit
Enter target directory for SDK (default: /usr/local/arago-2013.05): ./linux-devkit
You are about to install the SDK to "/home/tl/ti/mcsdk_1_01_00_02/linux-devkit". Proceed
[Y/n]?y
Extracting SDK...done
Setting it up...done
SDK has been successfully set up and is ready to be used.
--------------------------------------------------------------------------------
TISDK setup completed!
Please continue reading the Software Developer's Guide for more information on
how to develop software on the EVM
--------------------------------------------------------------------------------
```

图 23

最后看到"TI SDK setup completed!"，说明设置已经完成。

备注：如果期间提示，需要安装缺失的组件，根据提示，按"y"安装即可。由于之后不使用 MCSDK 的 U-Boot、内核、文件系统，因此以上部分相关设置可以忽视。

4 syslink 配置、编译、安装

安装 MCSDK 时，会将自动将 syslink 安装在相同的目录下，下文将介绍 syslink 配置、编译和示例演示。在开始 syslink 编译之前，请确保以下几点：

（1） 已安装 arm-none-linux- gnueabi-gcc-4.3.3 交叉编译工具链。

（2） 内核源码正确编译。

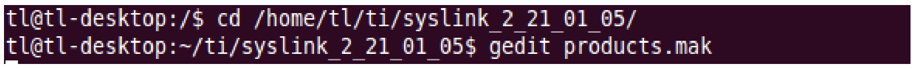
（3） 文件系统正确解压在 Ubuntu 虚拟机。

5 配置 syslink

进入"/home/tl/ti/syslink_2_21_01_05"，打开配置文件 products.mak。

Host#　cd /home/tl/ti/syslink_2_21_01_05

Host#　gedit products.mak



```
tl@tl-desktop:/$ cd /home/tl/ti/syslink_2_21_01_05/
tl@tl-desktop:~/ti/syslink_2_21_01_05$ gedit products.mak
```

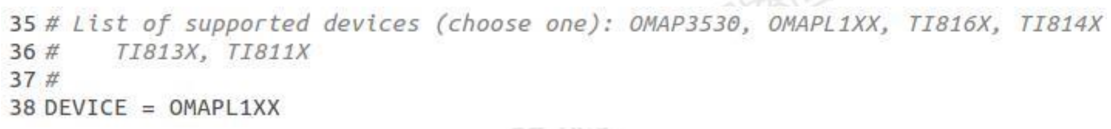
图 24

修改如下地方：

备注：由于配置容易出错，已将配置文件 product.mak 放在光盘 shell 目录下，可以将此文件覆盖"/home/tl/ti/syslink_2_21_01_05/products.mak"，然后再根据个人的实际情况小修改即可。

（1）　DEVICE = _your_device_

改为　DEVICE = OMAPL1XX　　//表示编译 OMAPL138

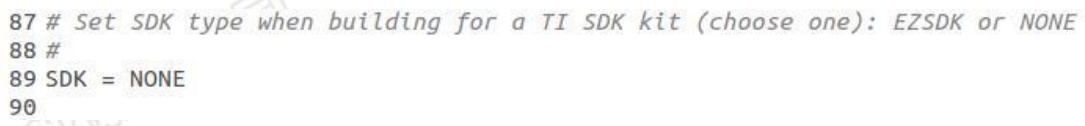


```
# List of supported devices (choose one): OMAP3530, OMAPL1XX, TI816X, TI814X
#     TI813X, TI811X
#
DEVICE = OMAPL1XX
```

图 25

（2）　SDK = _your_sdk_

改为　SDK = NONE　　//SDK 类型为 NONE

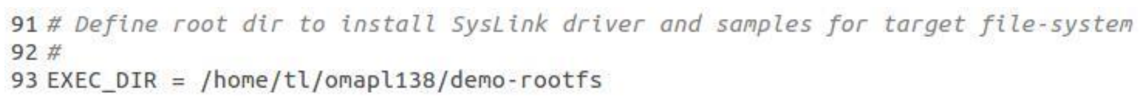


```
# Set SDK type when building for a TI SDK kit (choose one): EZSDK or NONE
#
SDK = NONE

```

图 26

（3） EXEC_DIR = _your_filesys_

改为 EXEC_DIR = /home/tl/omapl138/demo-rootfs //syslink 驱动和演示程序安装路径，一般设置为 nfs 或者 SD 卡的文件系统

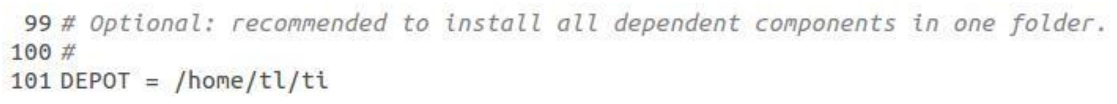


```
# Define root dir to install SysLink driver and samples for target file-system
#
EXEC_DIR = /home/tl/omapl138/demo-rootfs
```

图 27

（4） DEPOT = _your_depot_folder_

改为 DEPOT = /home/tl/ti //MCSDK 的安装路径



```
# Optional: recommended to install all dependent components in one folder.
#
DEPOT = /home/tl/ti
```

图 28

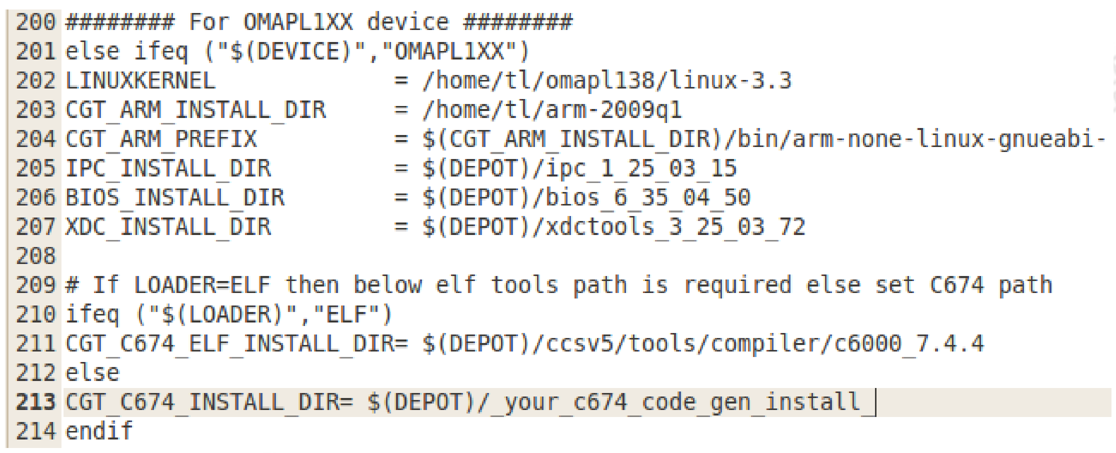
（5） 将以下内容：

```
######## For OMAPL1XX device ########
else ifeq ("$(DEVICE)","OMAPL1XX")
LINUXKERNEL              = $(DEPOT)/_your_linux_kernel_install_
CGT_ARM_INSTALL_DIR      = $(DEPOT)/_your_arm_code_gen_install_
CGT_ARM_PREFIX           = $(CGT_ARM_INSTALL_DIR)/bin/arm-none-linux-gnueabi-
IPC_INSTALL_DIR          = $(DEPOT)/_your_ipc_install_
BIOS_INSTALL_DIR         = $(DEPOT)/_your_bios_install_
XDC_INSTALL_DIR          = $(DEPOT)/_your_xdctools_install_

# If LOADER=ELF then below elf tools path is required else set C674 path
ifeq ("$(LOADER)","ELF")
```

```
CGT_C674_ELF_INSTALL_DIR  = $(DEPOT)/_your_c674elf_code_gen_install_
else
CGT_C674_INSTALL_DIR      = $(DEPOT)/_your_c674_code_gen_install_
endif
```

改为

```
######## For OMAPL1XX device ########
else ifeq ("$(DEVICE)","OMAPL1XX")
LINUXKERNEL               = /home/tl/omapl138/linux-3.3     //内核源码路径
CGT_ARM_INSTALL_DIR       = /home/tl/arm-2009q1     //交叉编译工具链安装路径
CGT_ARM_PREFIX                = $(CGT_ARM_INSTALL_DIR)/bin/arm-none-linux-gnueabi-
IPC_INSTALL_DIR           = $(DEPOT)/ipc_1_25_03_15     //ipc 安装路径
BIOS_INSTALL_DIR          = $(DEPOT)/bios_6_35_04_50     //bios 安装路径
XDC_INSTALL_DIR           = $(DEPOT)/xdctools_3_25_03_72     //xdc 安装路径

# If LOADER=ELF then below elf tools path is required else set C674 path
ifeq ("$(LOADER)","ELF")
CGT_C674_ELF_INSTALL_DIR  = $(DEPOT)/ccsv5/tools/compiler/c6000_7.4.4     //dsp 编译
器路径
else
CGT_C674_INSTALL_DIR      = $(DEPOT)/_your_c674_code_gen_install_
endif
```



```
######## For OMAPL1XX device ########
else ifeq ("$(DEVICE)","OMAPL1XX")
LINUXKERNEL                = /home/tl/omapl138/linux-3.3
CGT_ARM_INSTALL_DIR        = /home/tl/arm-2009q1
CGT_ARM_PREFIX             = $(CGT_ARM_INSTALL_DIR)/bin/arm-none-linux-gnueabi-
IPC_INSTALL_DIR            = $(DEPOT)/ipc_1_25_03_15
BIOS_INSTALL_DIR           = $(DEPOT)/bios_6_35_04_50
XDC_INSTALL_DIR            = $(DEPOT)/xdctools_3_25_03_72

# If LOADER=ELF then below elf tools path is required else set C674 path
ifeq ("$(LOADER)","ELF")
CGT_C674_ELF_INSTALL_DIR= $(DEPOT)/ccsv5/tools/compiler/c6000_7.4.4
else
CGT_C674_INSTALL_DIR= $(DEPOT)/_your_c674_code_gen_install_
endif
```

图 29

配置完成后，保存退出。

6 编译 syslink 源码

编译 syslink 之前，先将以下两个宏定义添加到 syslink 中的 Omapl1xxIpcInt.c、omapl1xx_phy_shmem.c、omapl1xxpwr.c 文件开头，否则编译会出错。

#undef __ASM_ARCH_HARDWARE_H

#include <mach/hardware.h>

以上三个文件的路径是：

（1） /home/tl/ti/syslink_2_21_01_05/packages/ti/syslink/ipc/hlos/knl/notifyDrivers/arch/omapl1xx/Omapl1xxIpcInt.c

（2） /home/tl/ti/syslink_2_21_01_05/packages/ti/syslink/family/hlos/knl/omapl1xx/omapl1xxdsp/Linux/omapl1xx_phy_shmem.c

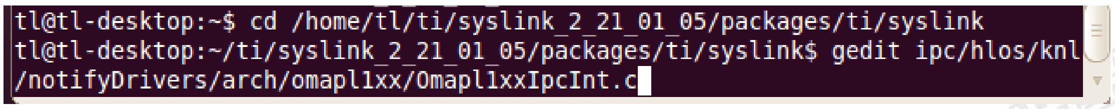
（3） /home/tl/ti/syslink_2_21_01_05/packages/ti/syslink/family/hlos/knl/omapl1xx/omapl1xxdsp/omapl1xxpwr.c

➢ 修改 Omapl1xxIpcInt.c

执行以下命令修改：

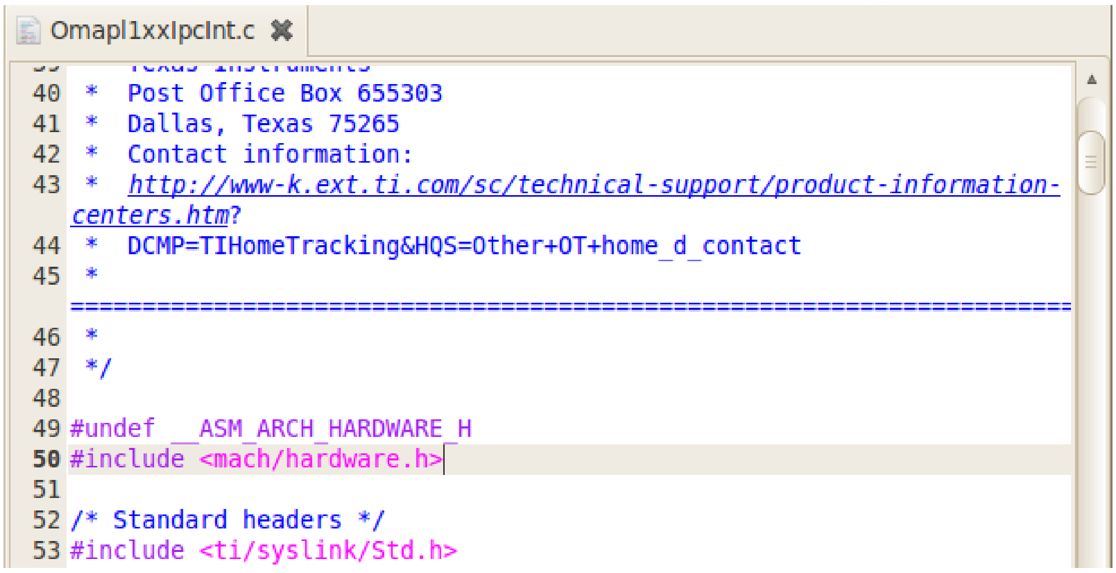
Host# cd /home/tl/ti/syslink_2_21_01_05/packages/ti/syslink

Host# gedit ipc/hlos/knl/notifyDrivers/arch/omapl1xx/Omapl1xxIpcInt.c



```
tl@tl-desktop:~$ cd /home/tl/ti/syslink_2_21_01_05/packages/ti/syslink
tl@tl-desktop:~/ti/syslink_2_21_01_05/packages/ti/syslink$ gedit ipc/hlos/knl
/notifyDrivers/arch/omapl1xx/Omapl1xxIpcInt.c
```

图 30



```
Omapl1xxIpcInt.c
 *  Post Office Box 655303
 *  Dallas, Texas 75265
 *  Contact information:
 *  http://www-k.ext.ti.com/sc/technical-support/product-information-
centers.htm?
 *  DCMP=TIHomeTracking&HQS=Other+OT+home_d_contact
 *
===========================================================================
 *
 */

#undef __ASM_ARCH_HARDWARE_H
#include <mach/hardware.h>

/* Standard headers */
#include <ti/syslink/Std.h>
```

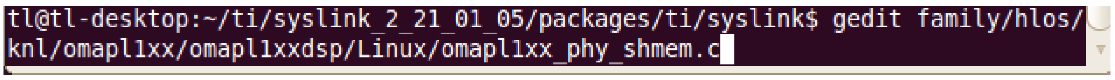
图 31

添加内容后，保存退出。

➢ 修改 omapl1xx_phy_shmem.c

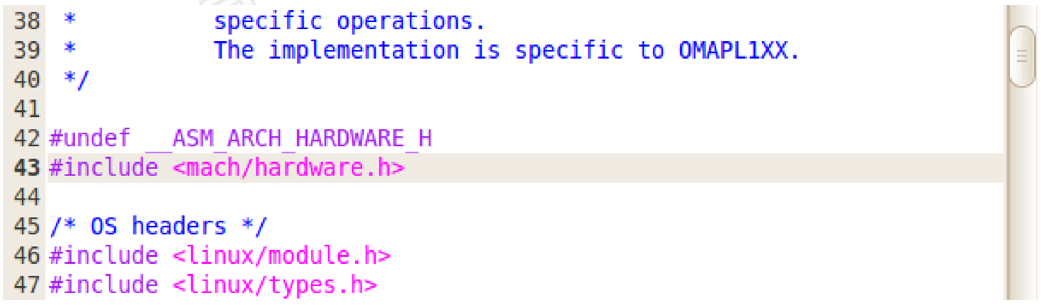
在当前路径下，执行以下命令：

Host# gedit family/hlos/knl/omapl1xx/omapl1xxdsp/Linux/omapl1xx_phy_shmem.c



```
tl@tl-desktop:~/ti/syslink_2_21_01_05/packages/ti/syslink$ gedit family/hlos/
knl/omapl1xx/omapl1xxdsp/Linux/omapl1xx_phy_shmem.c
```

图 32



```
 *          specific operations.
 *          The implementation is specific to OMAPL1XX.
 */

#undef __ASM_ARCH_HARDWARE_H
#include <mach/hardware.h>

/* OS headers */
#include <linux/module.h>
#include <linux/types.h>
```

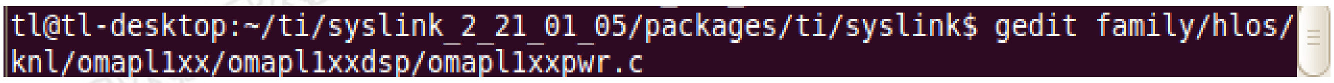
图 33

添加内容后，保存退出。

➢ 修改 omapl1xxpwr.c

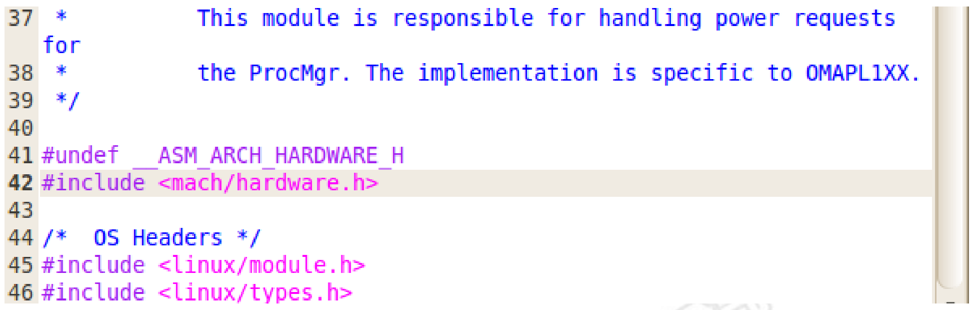
在当前路径下，执行以下命令：

Host# gedit family/hlos/knl/omapl1xx/omapl1xxdsp/omapl1xxpwr.c



```
tl@tl-desktop:~/ti/syslink_2_21_01_05/packages/ti/syslink$ gedit family/hlos/knl/omapl1xx/omapl1xxdsp/omapl1xxpwr.c
```

图 34



```
 *          This module is responsible for handling power requests
for
 *          the ProcMgr. The implementation is specific to OMAPL1XX.
 */

#undef __ASM_ARCH_HARDWARE_H
#include <mach/hardware.h>

/*  OS Headers */
#include <linux/module.h>
#include <linux/types.h>
```

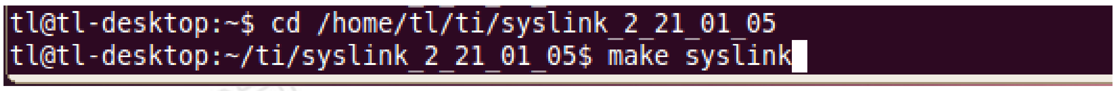
图 35

添加内容后，保存退出。

接下来开始编译 syslink，执行以下命令：

Host# cd /home/tl/ti/syslink_2_21_01_05

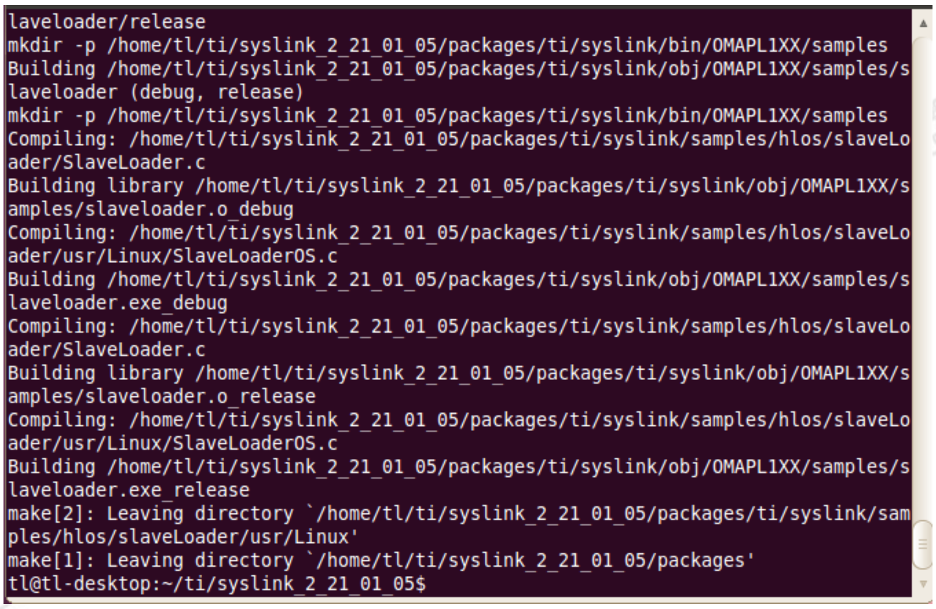
Host# make syslink



```
tl@tl-desktop:~$ cd /home/tl/ti/syslink_2_21_01_05
tl@tl-desktop:~/ti/syslink_2_21_01_05$ make syslink
```

图 36

编译成功如下图所示：



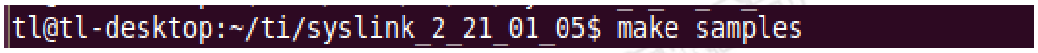
```
laveloader/release
mkdir -p /home/tl/ti/syslink_2_21_01_05/packages/ti/syslink/bin/OMAPL1XX/samples
Building /home/tl/ti/syslink_2_21_01_05/packages/ti/syslink/obj/OMAPL1XX/samples/s
laveloader (debug, release)
mkdir -p /home/tl/ti/syslink_2_21_01_05/packages/ti/syslink/bin/OMAPL1XX/samples
Compiling: /home/tl/ti/syslink_2_21_01_05/packages/ti/syslink/samples/hlos/slaveLo
ader/SlaveLoader.c
Building library /home/tl/ti/syslink_2_21_01_05/packages/ti/syslink/obj/OMAPL1XX/s
amples/slaveloader.o_debug
Compiling: /home/tl/ti/syslink_2_21_01_05/packages/ti/syslink/samples/hlos/slaveLo
ader/usr/Linux/SlaveLoaderOS.c
Building /home/tl/ti/syslink_2_21_01_05/packages/ti/syslink/obj/OMAPL1XX/samples/s
laveloader.exe_debug
Compiling: /home/tl/ti/syslink_2_21_01_05/packages/ti/syslink/samples/hlos/slaveLo
ader/SlaveLoader.c
Building library /home/tl/ti/syslink_2_21_01_05/packages/ti/syslink/obj/OMAPL1XX/s
amples/slaveloader.o_release
Compiling: /home/tl/ti/syslink_2_21_01_05/packages/ti/syslink/samples/hlos/slaveLo
ader/usr/Linux/SlaveLoaderOS.c
Building /home/tl/ti/syslink_2_21_01_05/packages/ti/syslink/obj/OMAPL1XX/samples/s
laveloader.exe_release
make[2]: Leaving directory `/home/tl/ti/syslink_2_21_01_05/packages/ti/syslink/sam
ples/hlos/slaveLoader/usr/Linux'
make[1]: Leaving directory `/home/tl/ti/syslink_2_21_01_05/packages'
tl@tl-desktop:~/ti/syslink_2_21_01_05$
```

图 37

7 编译 syslink 示例程序

在当前目录，执行以下命令：

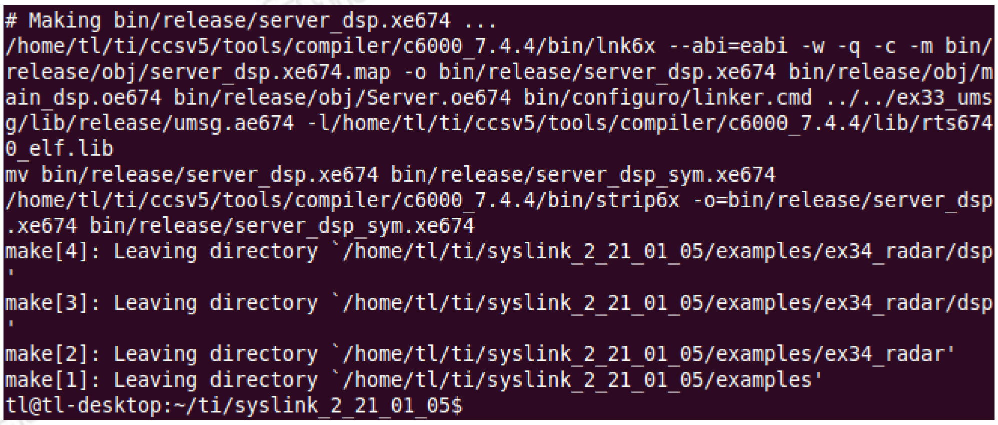
Host# make samples



```
tl@tl-desktop:~/ti/syslink_2_21_01_05$ make samples
```

图 38

编译成功如下图所示：



```
# Making bin/release/server_dsp.xe674 ...
/home/tl/ti/ccsv5/tools/compiler/c6000_7.4.4/bin/lnk6x --abi=eabi -w -q -c -m bin/
release/obj/server_dsp.xe674.map -o bin/release/server_dsp.xe674 bin/release/obj/m
ain_dsp.oe674 bin/release/obj/Server.oe674 bin/configuro/linker.cmd ../../ex33_ums
g/lib/release/umsg.ae674 -l/home/tl/ti/ccsv5/tools/compiler/c6000_7.4.4/lib/rts674
0_elf.lib
mv bin/release/server_dsp.xe674 bin/release/server_dsp_sym.xe674
/home/tl/ti/ccsv5/tools/compiler/c6000_7.4.4/bin/strip6x -o=bin/release/server_dsp
.xe674 bin/release/server_dsp_sym.xe674
make[4]: Leaving directory `/home/tl/ti/syslink_2_21_01_05/examples/ex34_radar/dsp
'
make[3]: Leaving directory `/home/tl/ti/syslink_2_21_01_05/examples/ex34_radar/dsp
'
make[2]: Leaving directory `/home/tl/ti/syslink_2_21_01_05/examples/ex34_radar'
make[1]: Leaving directory `/home/tl/ti/syslink_2_21_01_05/examples'
tl@tl-desktop:~/ti/syslink_2_21_01_05$
```

图 39

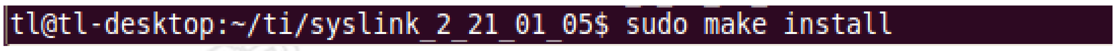
至此，整个 syslink 已经编译完成。

8 syslink 示例程序演示

8.1 安装 syslink 驱动和示例程序到文件系统

在当前目录，执行以下命令将 syslink 驱动和示例程序安装到文件系统：

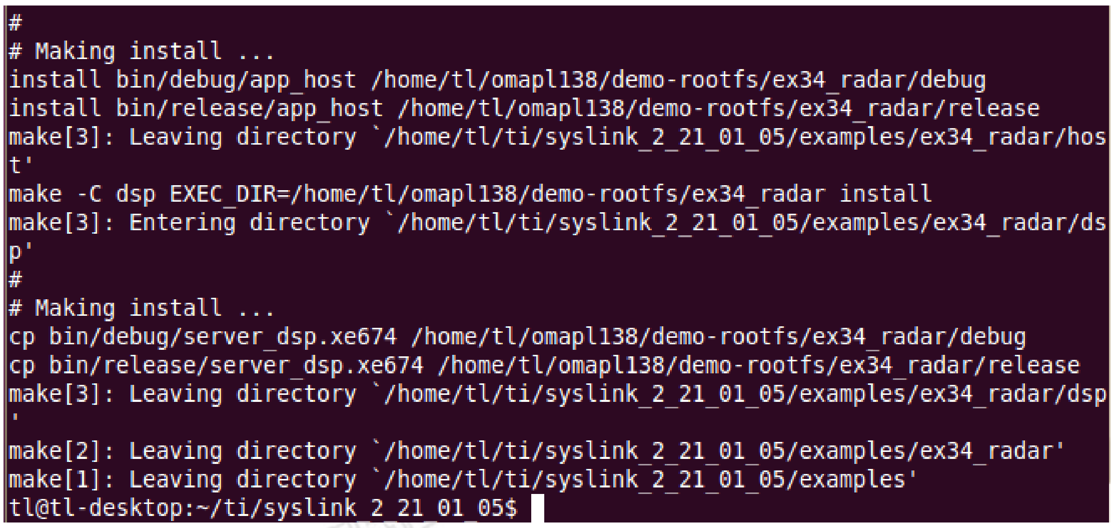
Host# sudo make install



```
tl@tl-desktop:~/ti/syslink_2_21_01_05$ sudo make install
```

图 40

安装成功如下图所示：



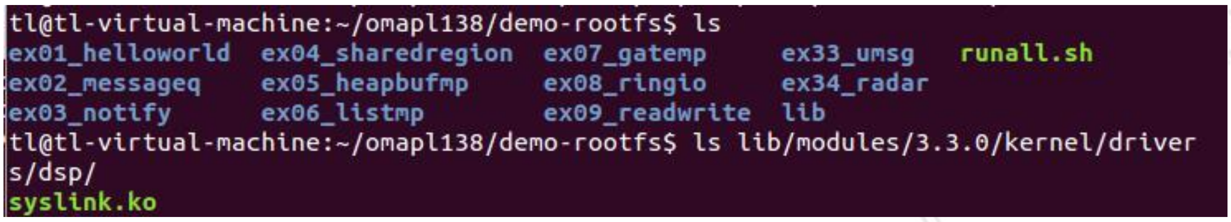
```
#
# Making install ...
install bin/debug/app_host /home/tl/omapl138/demo-rootfs/ex34_radar/debug
install bin/release/app_host /home/tl/omapl138/demo-rootfs/ex34_radar/release
make[3]: Leaving directory `/home/tl/ti/syslink_2_21_01_05/examples/ex34_radar/hos
t'
make -C dsp EXEC_DIR=/home/tl/omapl138/demo-rootfs/ex34_radar install
make[3]: Entering directory `/home/tl/ti/syslink_2_21_01_05/examples/ex34_radar/ds
p'
#
# Making install ...
cp bin/debug/server_dsp.xe674 /home/tl/omapl138/demo-rootfs/ex34_radar/debug
cp bin/release/server_dsp.xe674 /home/tl/omapl138/demo-rootfs/ex34_radar/release
make[3]: Leaving directory `/home/tl/ti/syslink_2_21_01_05/examples/ex34_radar/dsp
'
make[2]: Leaving directory `/home/tl/ti/syslink_2_21_01_05/examples/ex34_radar'
make[1]: Leaving directory `/home/tl/ti/syslink_2_21_01_05/examples'
tl@tl-desktop:~/ti/syslink_2_21_01_05$
```

图 41

执行以下命令查看是否已经安装了 syslink 驱动和示例程序。

Host# cd /home/tl/omapl138/demo-rootfs/

Host# ls lib/modules/3.3.0/kernel/drivers/dsp/



```
tl@tl-virtual-machine:~/omapl138/demo-rootfs$ ls
ex01_helloworld  ex04_sharedregion  ex07_gatemp     ex33_umsg   runall.sh
ex02_messageq    ex05_heapbufmp     ex08_ringio     ex34_radar
ex03_notify      ex06_listmp        ex09_readwrite  lib
tl@tl-virtual-machine:~/omapl138/demo-rootfs$ ls lib/modules/3.3.0/kernel/driver
s/dsp/
syslink.ko
```

图 42

可以看到在文件系统"lib/modules/3.3.0/kernel/drivers/dsp/"目录下有 syslink 驱动程序 syslink.ko 文件和文件系统根目录下有"ex**_##"的示例程序。

8.2 运行 syslink 示例程序

创龙提供的 U-Boot 会根据当前 CPU 型号和内存大小自动产生推荐的内存分配参数，并将此参数赋值给 mem_args 变量，U-Boot 默认使用 mem_args 变量值启动系统，下表为部分举例：

表 1

CPU 型号	内存大小	mem_args 变量值	备注
OMAP-L138	128MByte	mem=32M@0xc0000000 mem=64M@0xc4000000	DSP 使用 32MByte； ARM 使用 96MByte；

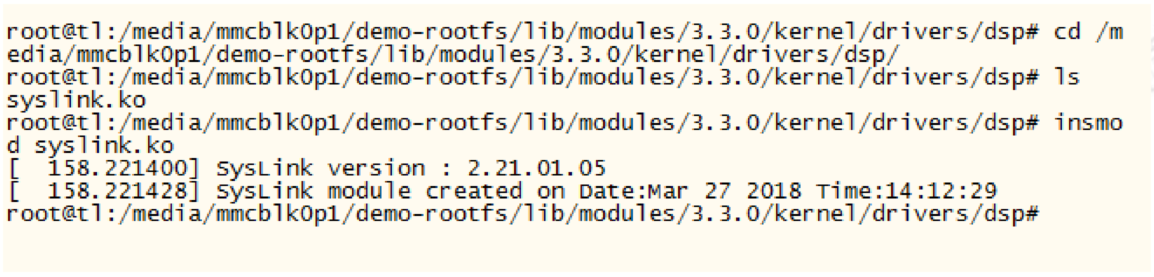
OMAP-L138	256Myte	mem=32M@0xc0000000
		mem=192M@0xc4000000
			DSP 使用 32MByte；
			ARM 使用 224MByte；

AM1808	128MByte	mem=128M@0xc0000000	
AM1808	256MByte	mem=256M@0xc0000000	
…	…	…	…

8.2.1 安装 syslink 驱动

将 syslink 的示例程序 demo-rootfs 文件夹拷贝到开发板中，启动开发板，进入文件系统的"demo-rootfs/lib/modules/3.3.0/kernel/drivers/dsp"路径下，执行命令安装 syslink 驱动：

Target# insmod syslink.ko



```
root@tl:/media/mmcblk0p1/demo-rootfs/lib/modules/3.3.0/kernel/drivers/dsp# cd /m
edia/mmcblk0p1/demo-rootfs/lib/modules/3.3.0/kernel/drivers/dsp/
root@tl:/media/mmcblk0p1/demo-rootfs/lib/modules/3.3.0/kernel/drivers/dsp# ls
syslink.ko
root@tl:/media/mmcblk0p1/demo-rootfs/lib/modules/3.3.0/kernel/drivers/dsp# insmo
d syslink.ko
[  158.221400] SysLink version : 2.21.01.05
[  158.221428] SysLink module created on Date:Mar 27 2018 Time:14:12:29
root@tl:/media/mmcblk0p1/demo-rootfs/lib/modules/3.3.0/kernel/drivers/dsp#
```

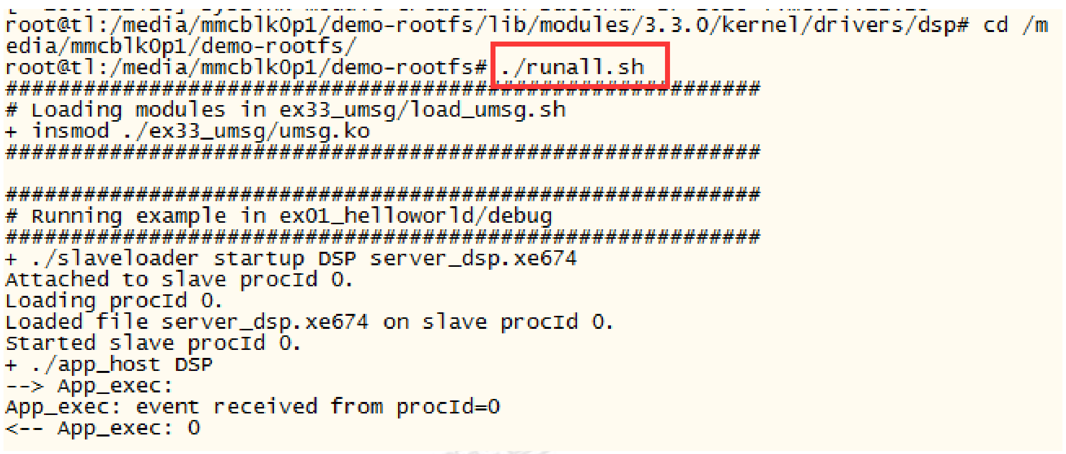
图 43

8.2.2 运行 syslink 示例程序

Target# cd /

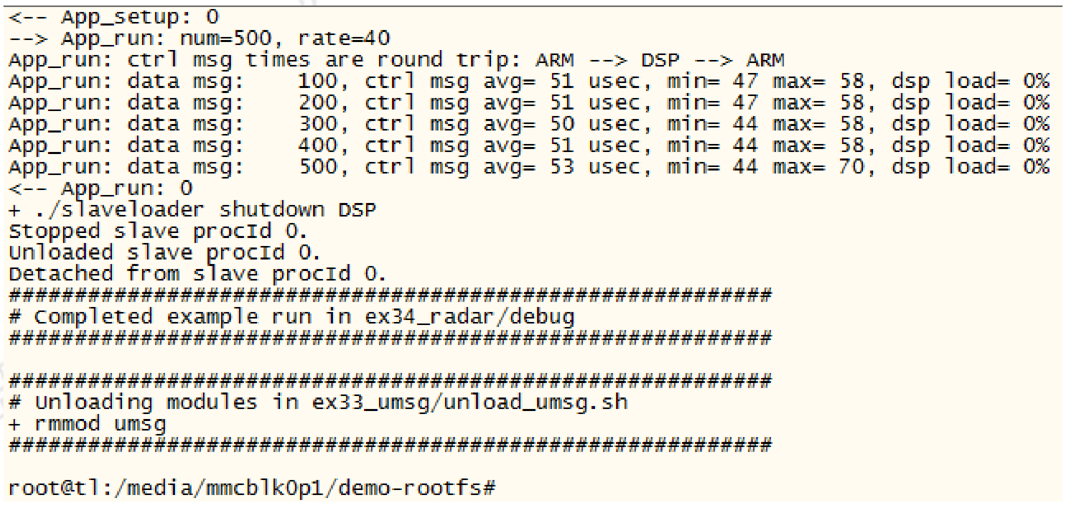
Target# ./runall.sh

成功运行如下图所示：



```
root@tl:/media/mmcblk0p1/demo-rootfs/lib/modules/3.3.0/kernel/drivers/dsp# cd /m
edia/mmcblk0p1/demo-rootfs/
root@tl:/media/mmcblk0p1/demo-rootfs# ./runall.sh
############################################################
# Loading modules in ex33_umsg/load_umsg.sh
+ insmod ./ex33_umsg/umsg.ko
############################################################

############################################################
# Running example in ex01_helloworld/debug
############################################################
+ ./slaveloader startup DSP server_dsp.xe674
Attached to slave procId 0.
Loading procId 0.
Loaded file server_dsp.xe674 on slave procId 0.
Started slave procId 0.
+ ./app_host DSP
--> App_exec:
App_exec: event received from procId=0
<-- App_exec: 0
```

图 44



```
<-- App_setup: 0
--> App_run: num=500, rate=40
App_run: ctrl msg times are round trip: ARM --> DSP --> ARM
App_run: data msg:    100, ctrl msg avg= 51 usec, min= 47 max= 58, dsp load= 0%
App_run: data msg:    200, ctrl msg avg= 51 usec, min= 47 max= 58, dsp load= 0%
App_run: data msg:    300, ctrl msg avg= 50 usec, min= 44 max= 58, dsp load= 0%
App_run: data msg:    400, ctrl msg avg= 51 usec, min= 44 max= 58, dsp load= 0%
App_run: data msg:    500, ctrl msg avg= 53 usec, min= 44 max= 70, dsp load= 0%
<-- App_run: 0
+ ./slaveloader shutdown DSP
Stopped slave procId 0.
Unloaded slave procId 0.
Detached from slave procId 0.
############################################################
# Completed example run in ex34_radar/debug
############################################################

############################################################
# Unloading modules in ex33_umsg/unload_umsg.sh
+ rmmod umsg
############################################################

root@tl:/media/mmcblk0p1/demo-rootfs#
```

图 45

8.3 syslink 示例程序解析

每个示例目录中有 readme.txt 和 run.sh 文件说明如何使用示例，而在开发板中运行 runall.sh 是运行了所有的 syslink 示例程。

备注：以下内容摘录于 http://blog.csdn.net/crushonme/article/details/10287693

Slaveloader

在 OMAPL138 的 ARM Linux 操作系统中，syslink 提供了 slaveloader 组件去加载、启动、停止 DSP 处理器，实现了对 DSP 核的管理，同时也是使用 slaveloader 组件去运行 syslink 示例程序。

运行"slaveloader"组件有四个参数：

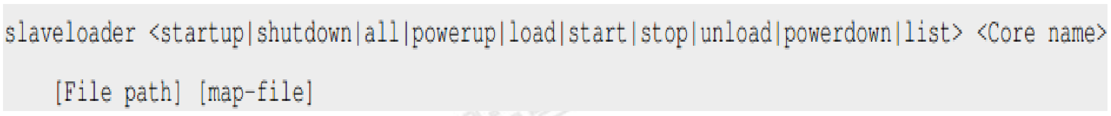
参数 1： startup|shutdown|all|powerup|load|start|stop|unload|powerdown|list

参数 2： Core name　//远程处理器名称，一般是 DSP

参数 3： File path　//可执行文件路径，当参数 2 为 startup/load/all/时必填

参数 4： map-file　//map 文件，当远程处理器 MMU 功能开启时必填

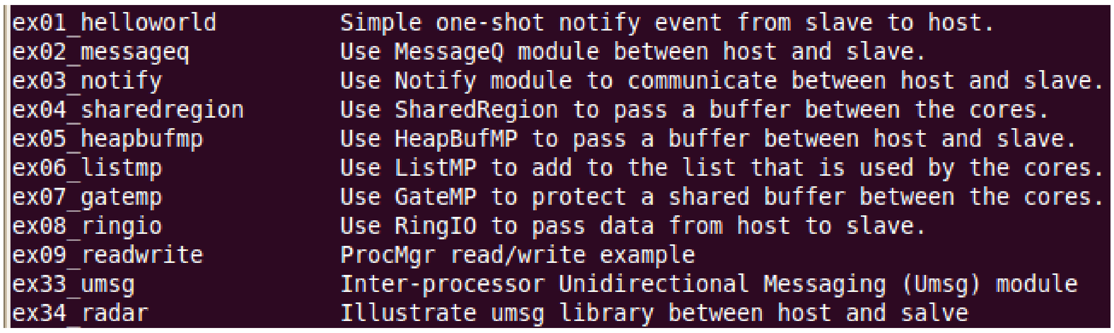
运行命令格式如下图所示：



```
slaveloader <startup|shutdown|all|powerup|load|start|stop|unload|powerdown|list> <Core name>
    [File path] [map-file]
```

图 46

可以通过各个 syslink 示例目录下的 run.sh 脚本查看使用 slaveloader 运行示例程序的具体方法。下图是各个示例的功能简介：



```
ex01_helloworld        Simple one-shot notify event from slave to host.
ex02_messageq          Use MessageQ module between host and slave.
ex03_notify            Use Notify module to communicate between host and slave.
ex04_sharedregion      Use SharedRegion to pass a buffer between the cores.
ex05_heapbufmp         Use HeapBufMP to pass a buffer between host and slave.
ex06_listmp            Use ListMP to add to the list that is used by the cores.
ex07_gatemp            Use GateMP to protect a shared buffer between the cores.
ex08_ringio            Use RingIO to pass data from host to slave.
ex09_readwrite         ProcMgr read/write example
ex33_umsg              Inter-processor Unidirectional Messaging (Umsg) module
ex34_radar             Illustrate umsg library between host and salve
```

图 47

备注：在单独运行各个示例程序前，务必先安装 syslink 驱动,安装命令：

Target# insmod /lib/modules/3.3.0/kernel/drivers/dsp/syslink.ko TRACE=1 TRACEFAILURE=1

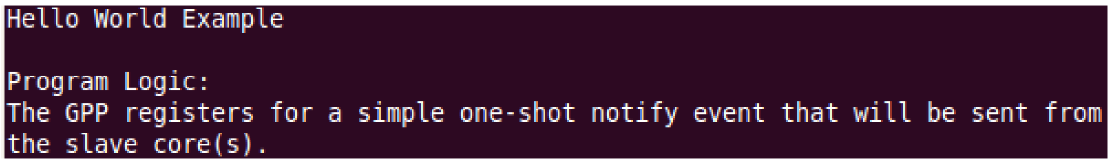
下面将针对每个示例进行解析。

8.3.1 ex01_helloworld

示例名字：helloworld

功能说明：GPP（ARM）端注册一个来自 DSP 端的简单一次性通知事件。

参考英文资料：



```
Hello World Example

Program Logic:
The GPP registers for a simple one-shot notify event that will be sent from
the slave core(s).
```

图 48

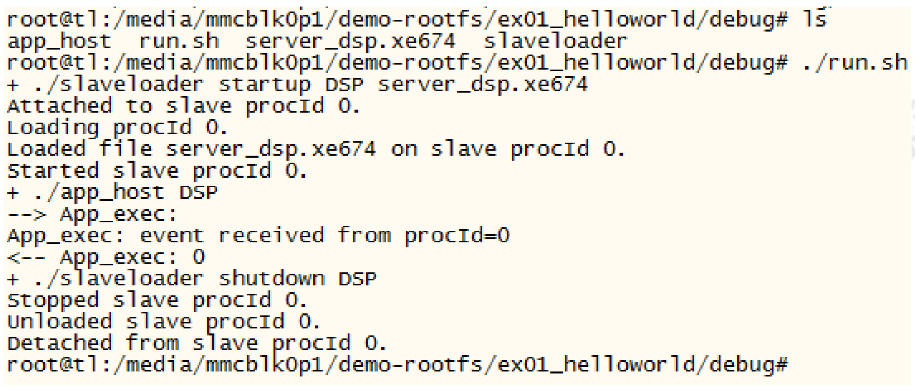
运行命令：

Target# cd /ex01_helloworld/debug

Target# ls

Target# ./run.sh

成功运行提示如下图：



```
root@tl:/media/mmcblk0p1/demo-rootfs/ex01_helloworld/debug# ls
app_host  run.sh  server_dsp.xe674  slaveloader
root@tl:/media/mmcblk0p1/demo-rootfs/ex01_helloworld/debug# ./run.sh
+ ./slaveloader startup DSP server_dsp.xe674
Attached to slave procId 0.
Loading procId 0.
Loaded file server_dsp.xe674 on slave procId 0.
Started slave procId 0.
+ ./app_host DSP
--> App_exec:
App_exec: event received from procId=0
<-- App_exec: 0
+ ./slaveloader shutdown DSP
Stopped slave procId 0.
Unloaded slave procId 0.
Detached from slave procId 0.
root@tl:/media/mmcblk0p1/demo-rootfs/ex01_helloworld/debug#
```

图 49

8.3.2 ex02_messageq

示例名字：MessageQ

功能说明：基于队列的消息传递，负责 GPP 与 DSP 端的可变长度的短消息交互。

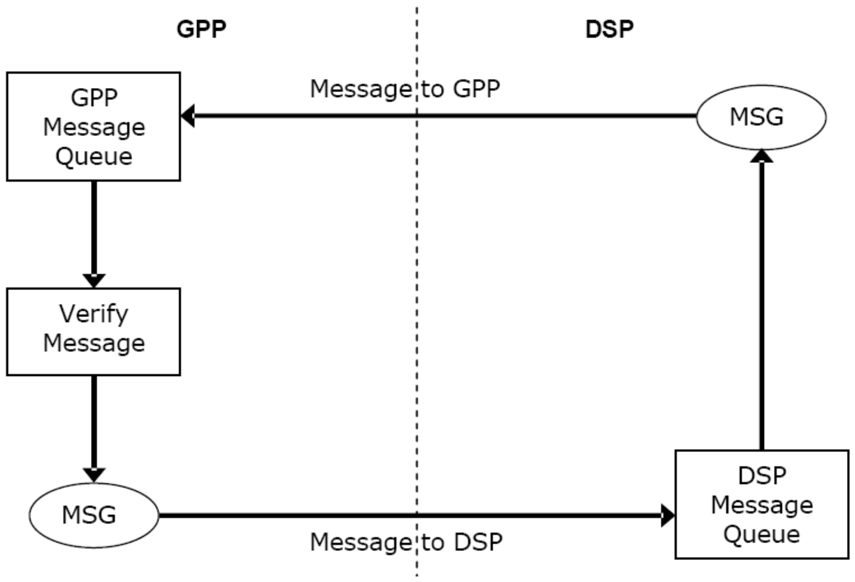
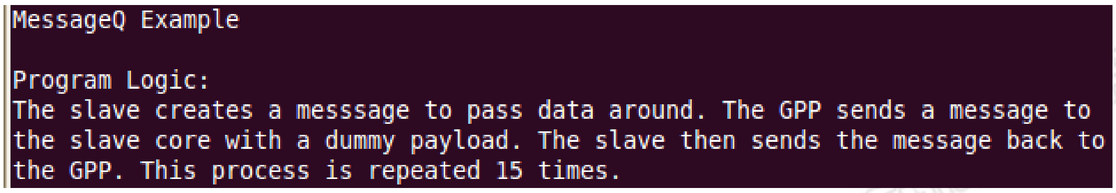


图 50

参考英文资料：



```
MessageQ Example

Program Logic:
The slave creates a messsage to pass data around. The GPP sends a message to
the slave core with a dummy payload. The slave then sends the message back to
the GPP. This process is repeated 15 times.
```

图 51

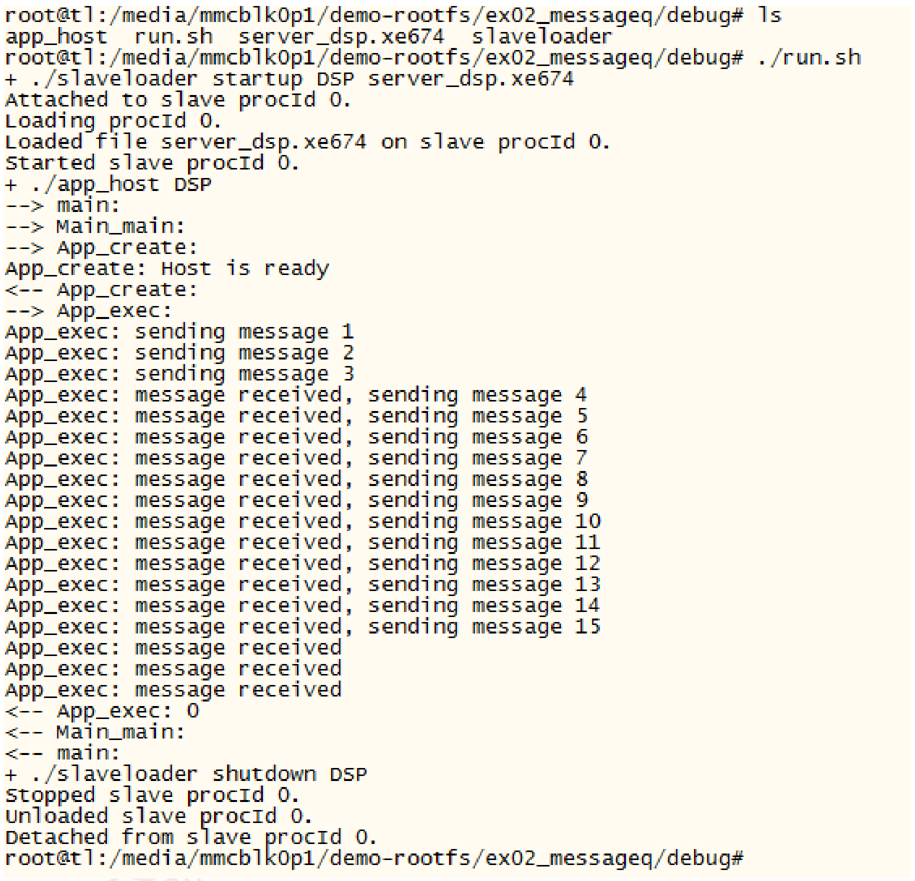
运行命令：

Target# cd /ex02_messageq/debug/

Target# ls

Target# ./run.sh

成功运行提示如下图：



```
root@tl:/media/mmcblk0p1/demo-rootfs/ex02_messageq/debug# ls
app_host  run.sh  server_dsp.xe674  slaveloader
root@tl:/media/mmcblk0p1/demo-rootfs/ex02_messageq/debug# ./run.sh
+ ./slaveloader startup DSP server_dsp.xe674
Attached to slave procId 0.
Loading procId 0.
Loaded file server_dsp.xe674 on slave procId 0.
Started slave procId 0.
+ ./app_host DSP
--> main:
--> Main_main:
--> App_create:
App_create: Host is ready
<-- App_create:
--> App_exec:
App_exec: sending message 1
App_exec: sending message 2
App_exec: sending message 3
App_exec: message received, sending message 4
App_exec: message received, sending message 5
App_exec: message received, sending message 6
App_exec: message received, sending message 7
App_exec: message received, sending message 8
App_exec: message received, sending message 9
App_exec: message received, sending message 10
App_exec: message received, sending message 11
App_exec: message received, sending message 12
App_exec: message received, sending message 13
App_exec: message received, sending message 14
App_exec: message received, sending message 15
App_exec: message received
App_exec: message received
App_exec: message received
<-- App_exec: 0
<-- Main_main:
<-- main:
+ ./slaveloader shutdown DSP
Stopped slave procId 0.
Unloaded slave procId 0.
Detached from slave procId 0.
root@tl:/media/mmcblk0p1/demo-rootfs/ex02_messageq/debug#
```

图 52

特点：

（1） 实现了处理期间变长消息的传递；

（2） 消息的传递都是通过操作消息队列来实现的；

（3） 每个消息队列可以有多个写者，但只能有一个读者；每个任务（task）可以对多个消息队列进行读写；

（4） 一个宿主在准备接收消息时，必须先创建消息队列，而在发送消息前，需要打开预定的接收消息队列；

常用在以下场景中：

（1） 在消息传递中有多个写者，但仅有一个读者；

（2） 所需要传递的消息超过 32bit，且长度可变；读写者的缓冲区大小相同；

（3） 处理期间需要频繁传递消息，在这种情况下，消息被依次放入队列，能保证不会丢消息；

（4） 消息队列为空时，调用 MessageQ_get()获取消息时会被阻塞，直到消息队列被写入消息；

（5） 支持处理器间移动消息队列，在这种情况下，调用 MessageQ_open()来定位队列位置，而消息传递部分代码不需要改动；

提供的 API 接口：

（1） **消息队列初始化：**MessageQ_Params_init()

（2） **消息队列创建/销毁：**MessageQ_create()/MessageQ_delete()，create 创建消息队列，并分配相应存储空间

（3） **消息队列打开/关闭：**MessageQ_open()/MessageQ_close()，open 时会返回远程处理器上的 QueID 的地址

（4） **为消息队列分配堆内存：**MessageQ_alloc()/MessageQ_free()

（5） **为消息队列注册/注销堆内存：**
MessageQ_registerHeap()/MessageQ_unregisterHeap()

（6） **向消息队列中放入/获取消息：**MessageQ_put()/MessageQ_get()

（7） **获取消息队列 ID：**MessageQ_getQueueId()

（8） **获取消息队列中消息数：**MessageQ_count()

（9） **在消息队列中嵌入消息：**MessageQ_setReplyQueue()

（10） **为消息队列解阻塞：** MessageQ_unblock()

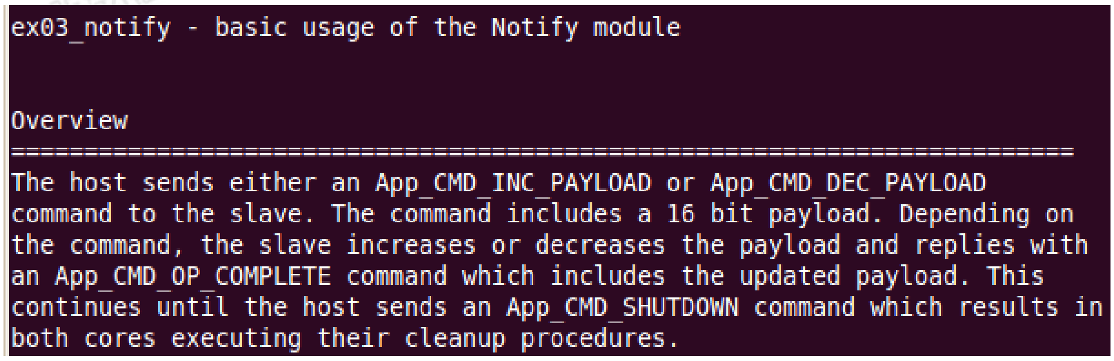
（11） **为调试消息队列加入 Trace：** MessageQ_setMsgTrace()

8.3.3 ex03_notify

示例名字： Notify

功能说明： 将硬件中断抽象成多组逻辑事件，是一种简单快捷的发送低于 32bit 信息的通信方式。

参考英文资料：



```
ex03_notify - basic usage of the Notify module


Overview
=========================================================================
The host sends either an App_CMD_INC_PAYLOAD or App_CMD_DEC_PAYLOAD
command to the slave. The command includes a 16 bit payload. Depending on
the command, the slave increases or decreases the payload and replies with
an App_CMD_OP_COMPLETE command which includes the updated payload. This
continues until the host sends an App_CMD_SHUTDOWN command which results in
both cores executing their cleanup procedures.
```

图 53

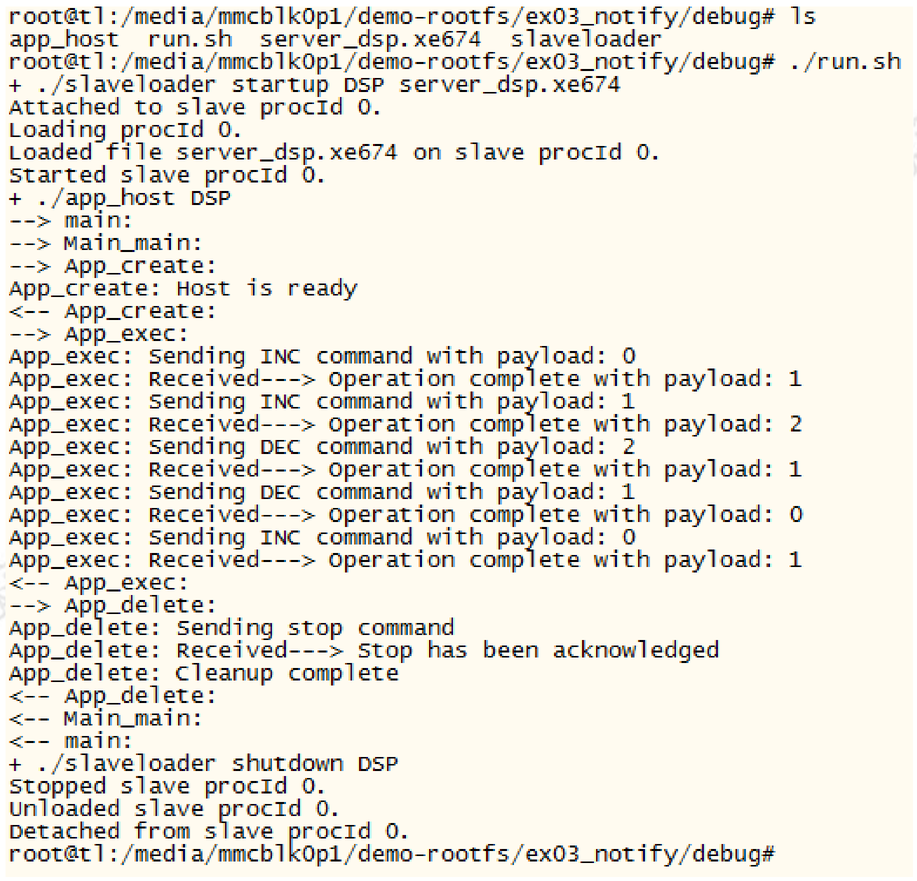
运行命令：

Target#　　cd /ex03_notify/debug/

Target#　　ls

Target#　　./run.sh

成功运行提示如下图：



```
root@tl:/media/mmcblk0p1/demo-rootfs/ex03_notify/debug# ls
app_host  run.sh  server_dsp.xe674  slaveloader
root@tl:/media/mmcblk0p1/demo-rootfs/ex03_notify/debug# ./run.sh
+ ./slaveloader startup DSP server_dsp.xe674
Attached to slave procId 0.
Loading procId 0.
Loaded file server_dsp.xe674 on slave procId 0.
Started slave procId 0.
+ ./app_host DSP
--> main:
--> Main_main:
--> App_create:
App_create: Host is ready
<-- App_create:
--> App_exec:
App_exec: Sending INC command with payload: 0
App_exec: Received---> Operation complete with payload: 1
App_exec: Sending INC command with payload: 1
App_exec: Received---> Operation complete with payload: 2
App_exec: Sending DEC command with payload: 2
App_exec: Received---> Operation complete with payload: 1
App_exec: Sending DEC command with payload: 1
App_exec: Received---> Operation complete with payload: 0
App_exec: Sending INC command with payload: 0
App_exec: Received---> Operation complete with payload: 1
<-- App_exec:
--> App_delete:
App_delete: Sending stop command
App_delete: Received---> Stop has been acknowledged
App_delete: Cleanup complete
<-- App_delete:
<-- Main_main:
<-- main:
+ ./slaveloader shutdown DSP
Stopped slave procId 0.
Unloaded slave procId 0.
Detached from slave procId 0.
root@tl:/media/mmcblk0p1/demo-rootfs/ex03_notify/debug#
```

图 54

特点：

（1）　同一个中断号可以注册多个事件，同一个事件可以有多个回调函数或者多个宿主（可以是处理器、线程或者任务），事件被触发后所有宿主都会被唤醒；

（2）　一个事件可以接收多个宿主发送来的通知（notification），事件所携带的参数最大支持 32bit；

（3）　事件是有优先级的，EventId 越小优先级越高，事件 0 的优先级最高，随着 EventId 增大优先级依次递减；当多个事件被触发，优先级最高的会最先响应；

（4）　Notify 模块使用硬件中断，因此不能被频繁调度。

提供的 API 接口：

（1）　**初始化并配置 Notify 组件：** Notify_attach();

（2）　**注册/注销事件：** Notify_registerEvent()/Notify_unregisterEvent()/

Notify_registerEventSingle()/Notify_unregisterEventSingle()

（3） **发送带参数的事件给某处理器：**Notify_sendEvent()

（4） **通过回调函数接收事件：**Notify_FnNotifyCbck()

（5） **使能/禁用事件：**Notify_disableEvent()/Notify_enableEvent()

（6） **其他逻辑接口：**Notify_eventAvailable()/Notify_intLineRegistered()/

Notify_numIntLines()/Notify_restore()

Notify 组件常用于传递附带消息少于 32bit 的场景，如信令传递、buffer 指针传递等。在信令传递时使用高优先级的事件，如事件 0。而在传递 buffer 指针是可以使用低优先级的事件，如事件 30 等。

在 Notify_sentEvent() API 中带有参数 waitClear，该参数为可选参数，如果 waitClear 为 TRUE，这就意味着多宿主事件无法及时响应，必须等待前一宿主事件结束后才能响应下一宿主；如果 waitClear 为 FALSE，最好不要为事件附带参数，否则多宿主事件可能会由于消息被覆盖而出现丢消息的现象。

该 API 最好不要在中断服务程序（ISR）调用（特别是 waitClear=TRUE 时），否则会导致中断调度出现异常（表现之一：高优先级的中断响应会延迟）。此外该 API 不能再使用 GateMP 模块锁保护的程序段中调用，否则可能会导致操作系统死锁。

由于其他模块使用了 Notify 机制，因此在 SysLink 中预留了部分事件号，这部分事件号用户需要慎重选用（如果你没有使用其他组建的话，可以考虑占用这部分事件号），在注册事件前可以使用 Notify_eventAvailable()来检查该事件是否可用，即该中断号上的该事件号是否被注册。

表格 1

Module	Event Ids
FrameQBufMgr	0
FrameQ	1
MessageQ(TransportShm)	2
RingIO	3

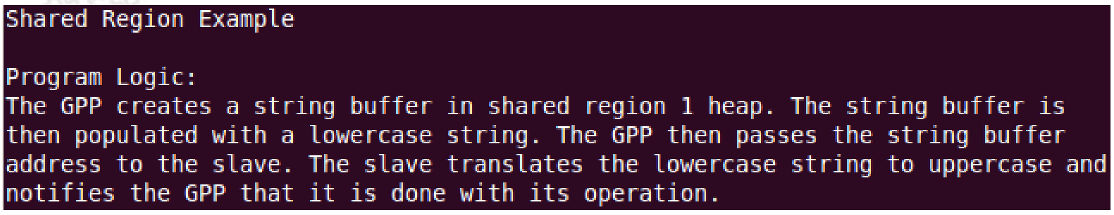
NameServerRemoteNotify	4

8.3.4 ex04_sharedregion

示例名字：SharedRegion

功能说明： SharedRegion 模块负责管理共享内存区。在一个有共享内存的多核架构中，普遍会遇到共享内存映射虚拟地址转换问题。

参考英文资料：



```
Shared Region Example

Program Logic:
The GPP creates a string buffer in shared region 1 heap. The string buffer is
then populated with a lowercase string. The GPP then passes the string buffer
address to the slave. The slave translates the lowercase string to uppercase and
notifies the GPP that it is done with its operation.
```

图 55

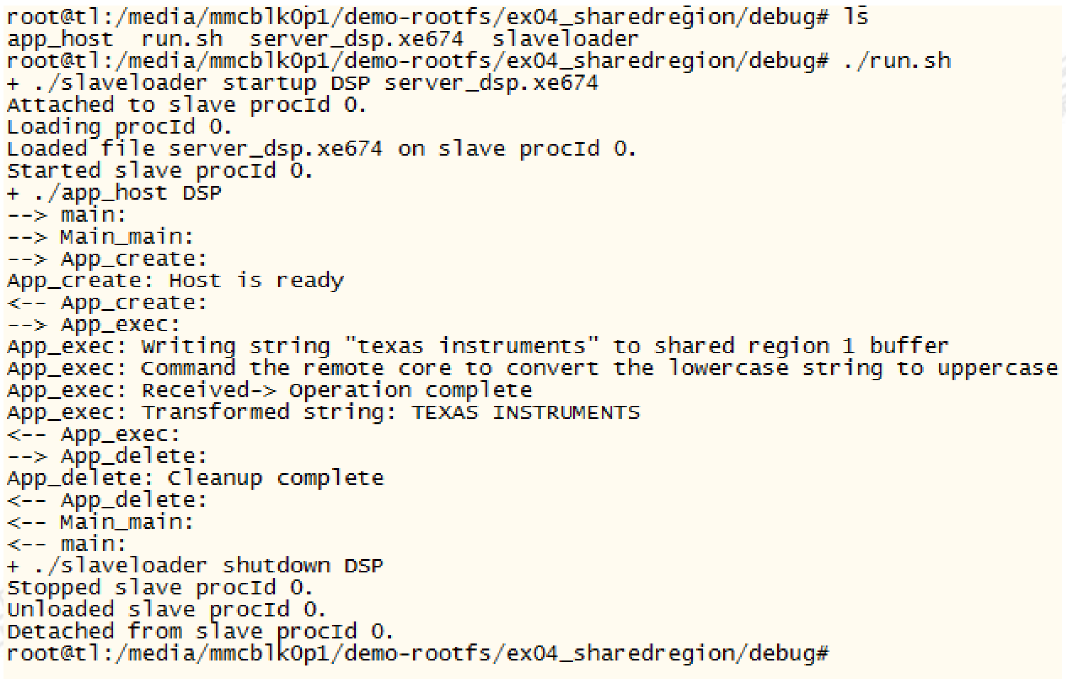
运行命令：

Target# cd /ex04_sharedregion/debug/

Target# ls

Target# ./run.sh

成功运行提示如下图：



```
root@tl:/media/mmcblk0p1/demo-rootfs/ex04_sharedregion/debug# ls
app_host  run.sh  server_dsp.xe674  slaveloader
root@tl:/media/mmcblk0p1/demo-rootfs/ex04_sharedregion/debug# ./run.sh
+ ./slaveloader startup DSP server_dsp.xe674
Attached to slave procId 0.
Loading procId 0.
Loaded file server_dsp.xe674 on slave procId 0.
Started slave procId 0.
+ ./app_host DSP
--> main:
--> Main_main:
--> App_create:
App_create: Host is ready
<-- App_create:
--> App_exec:
App_exec: Writing string "texas instruments" to shared region 1 buffer
App_exec: Command the remote core to convert the lowercase string to uppercase
App_exec: Received-> Operation complete
App_exec: Transformed string: TEXAS INSTRUMENTS
<-- App_exec:
--> App_delete:
App_delete: Cleanup complete
<-- App_delete:
<-- Main_main:
<-- main:
+ ./slaveloader shutdown DSP
Stopped slave procId 0.
Unloaded slave procId 0.
Detached from slave procId 0.
root@tl:/media/mmcblk0p1/demo-rootfs/ex04_sharedregion/debug#
```

图 56

提供的 API 接口：

（1） SharedRegion_clearEntry()

（2） SharedRegion_entryInit()

（3） SharedRegion_getCacheLineSize()

（4） SharedRegion_getEntry()、SharedRegion_setEntry()

（5） SharedRegion_getHeap()

（6） SharedRegion_getId()

（7） SharedRegion_getIdByName()

（8） SharedRegion_getNumRegions()

（9） SharedRegion_getPtr()

（10） SharedRegion_getSRPtr()

（11） SharedRegion_isCacheEnabled()

（12） SharedRegion_translateEnabled()

一般来说配置一个 SharedRegion 需要关心以下几个参数：

（1） **base**：The base address，共享内存区的基地址，这个所谓的基地址实际上是映射后的虚拟地址，并非物理地址；

（2） **len**：The length，共享内存区的大小，对于同一片共享内存，其所有者的查找表中该项值应该是相同的；

（3） **name**：The name of the region，该共享内存区的名字；

（4） **isValid**：Whether the region is valid，对于该处理器而言，是否具有权限去访问该共享内存区；

（5） **ownerProcId**：The id of the processor which owns the region，管理该内存区的处理器 ID，该处理器具有创建 HeapMemMP 的权限，而其他处理器只有使用的权限；

（6） **cacheEnable**：Whether the region is cacheable，是否为该共享内存区创建 cache；

（7） **cacheLineSize**：The cache line size，cache 的大小；

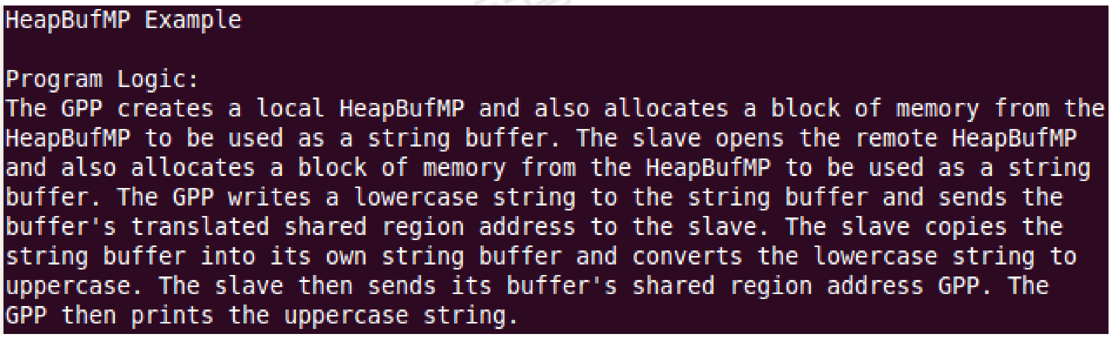
（8） **createHeap**：Whether a heap is created for the region，是否使用 Heap（堆）管理该内存区域；

8.3.5 ex05_heapbufmp

示例名字：HeapBufMP

功能说明：为用户提供了固定大小的缓冲池管理接口。

参考英文资料：



```
HeapBufMP Example

Program Logic:
The GPP creates a local HeapBufMP and also allocates a block of memory from the
HeapBufMP to be used as a string buffer. The slave opens the remote HeapBufMP
and also allocates a block of memory from the HeapBufMP to be used as a string
buffer. The GPP writes a lowercase string to the string buffer and sends the
buffer's translated shared region address to the slave. The slave copies the
string buffer into its own string buffer and converts the lowercase string to
uppercase. The slave then sends its buffer's shared region address GPP. The
GPP then prints the uppercase string.
```

图 57

运行命令：

Target# cd /ex05_heapbufmp/debug/

Target# ls

Target# ./run.sh

成功运行提示如下图：

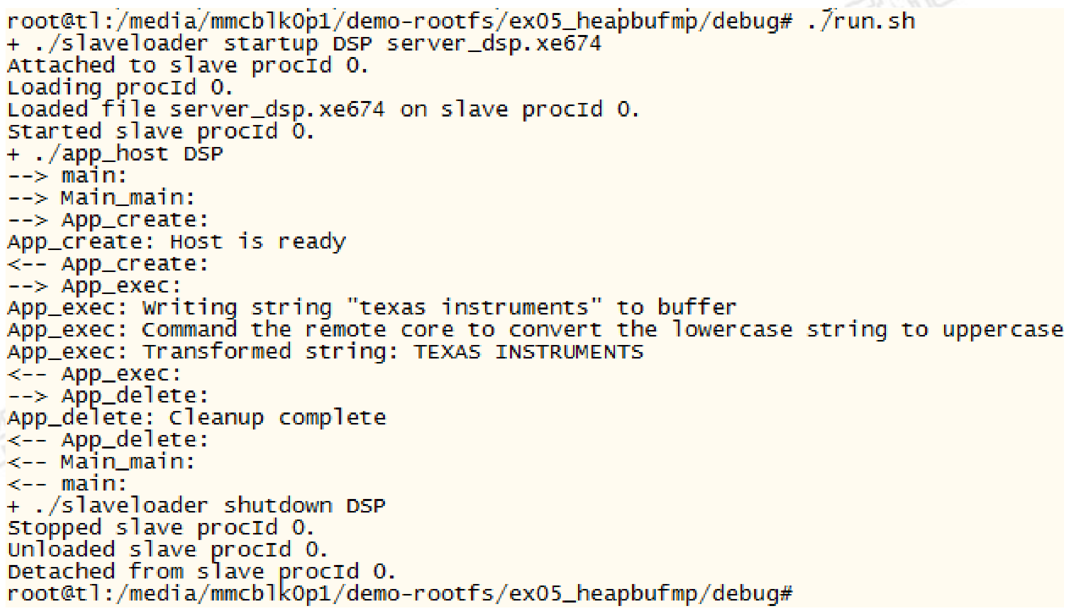

```
root@tl:/media/mmcblk0p1/demo-rootfs/ex05_heapbufmp/debug# ./run.sh
+ ./slaveloader startup DSP server_dsp.xe674
Attached to slave procId 0.
Loading procId 0.
Loaded file server_dsp.xe674 on slave procId 0.
Started slave procId 0.
+ ./app_host DSP
--> main:
--> Main_main:
--> App_create:
App_create: Host is ready
<-- App_create:
--> App_exec:
App_exec: Writing string "texas instruments" to buffer
App_exec: Command the remote core to convert the lowercase string to uppercase
App_exec: Transformed string: TEXAS INSTRUMENTS
<-- App_exec:
--> App_delete:
App_delete: Cleanup complete
<-- App_delete:
<-- Main_main:
<-- main:
+ ./slaveloader shutdown DSP
Stopped slave procId 0.
Unloaded slave procId 0.
Detached from slave procId 0.
root@tl:/media/mmcblk0p1/demo-rootfs/ex05_heapbufmp/debug#
```

图 58

提供的 API 接口：

（1） **HeapBufMP 创建/删除：**HeapBufMP_create()/HeapBufMP_delete()

（2） **HeapBufMP 打开/关闭：**HeapBufMP_open()/HeapBufMP_close()

（3） **HeapBufMP 参数初始化：**HeapBufMP_Params_init()

（4） **HeapBufMP 分配/释放内存：**HeapBufMP_alloc()/HeapBufMP_free()

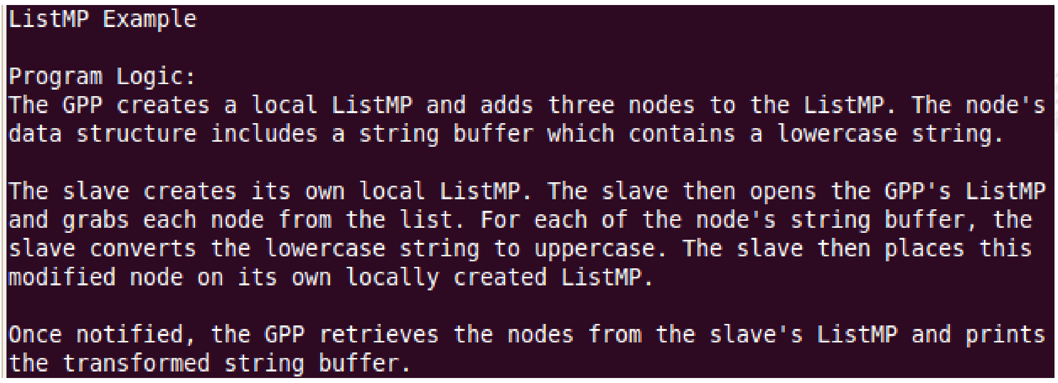
（5） **HeapBufMP 获取所有状态：**HeapBufMP_getExtendedStats()/ HeapBufMP_getStats()

8.3.6 ex06_listmp

示例名字：ListMP

功能说明：实现了多宿主双向循环链表，即该双向循环链表为多个处理器共同拥有，可以由多个处理器共同维护，共同使用。

参考英文资料：



```
ListMP Example

Program Logic:
The GPP creates a local ListMP and adds three nodes to the ListMP. The node's
data structure includes a string buffer which contains a lowercase string.

The slave creates its own local ListMP. The slave then opens the GPP's ListMP
and grabs each node from the list. For each of the node's string buffer, the
slave converts the lowercase string to uppercase. The slave then places this
modified node on its own locally created ListMP.

Once notified, the GPP retrieves the nodes from the slave's ListMP and prints
the transformed string buffer.
```

图 59

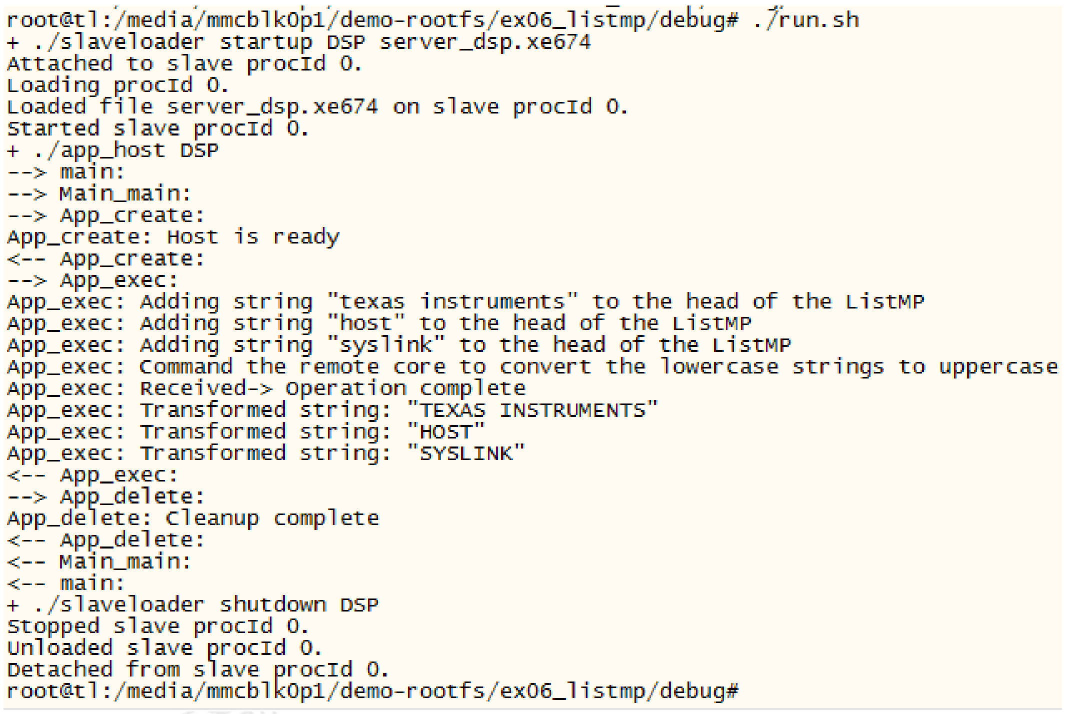
运行命令：

Target#　　cd /ex06_listmp/debug/

Target#　　ls

Target#　　./run.sh

成功运行提示如下图：



```
root@tl:/media/mmcblk0p1/demo-rootfs/ex06_listmp/debug# ./run.sh
+ ./slaveloader startup DSP server_dsp.xe674
Attached to slave procId 0.
Loading procId 0.
Loaded file server_dsp.xe674 on slave procId 0.
Started slave procId 0.
+ ./app_host DSP
--> main:
--> Main_main:
--> App_create:
App_create: Host is ready
<-- App_create:
--> App_exec:
App_exec: Adding string "texas instruments" to the head of the ListMP
App_exec: Adding string "host" to the head of the ListMP
App_exec: Adding string "syslink" to the head of the ListMP
App_exec: Command the remote core to convert the lowercase strings to uppercase
App_exec: Received-> Operation complete
App_exec: Transformed string: "TEXAS INSTRUMENTS"
App_exec: Transformed string: "HOST"
App_exec: Transformed string: "SYSLINK"
<-- App_exec:
--> App_delete:
App_delete: Cleanup complete
<-- App_delete:
<-- Main_main:
<-- main:
+ ./slaveloader shutdown DSP
Stopped slave procId 0.
Unloaded slave procId 0.
Detached from slave procId 0.
root@tl:/media/mmcblk0p1/demo-rootfs/ex06_listmp/debug#
```

图 60

特点：

ListMP 的实现区别于一般的双向循环链表，因此它不仅具有双向循环链表的特性外，还增添了其他的特性，比如以下几点：

（1） 实现了简单的多宿主协议，支持多个读写者（multi-reader、multi-writee）；
（2） 使用 Gate 作为内部保护机制，防止多个宿主处理器同时访问该链表；

ListMP 的实现并未加入通知机制，如果需要的话，可以在外部封装时引入 Notify 机制来实现；使用 ListMP 机制来管理的 buffers 都需要从共享内存区分配，包括从堆内存分配的 buffers 以及动态分配的内存。

常用在以下场景中：

（1） 需要被多个宿主访问并且需要频繁传递消息或者数据；
（2） 可用于无规则的消息传递，基于链表实现，因此读者可以遍历所有对象，并选出需要的对象进行处理；如果硬件支持快速队列，则无法完成队列遍历操作；
（3） 可以自定义消息优先级，同样是基于链表实现，读者可以随意的选择在链表头部还是链表的尾部来插入消息或者实现链表对象的位置调整，进而实现消息的优先级选择；如果硬件支持快速队列，则无法完成队列遍历操作；
（4） 无内置通知机制，可以灵活的外部通知机制来实现。譬如根据实际情况，选用 Notify 来实现，亦或是使用选用 MessageQ 则可以使用最少的中断资源实现性能优良的通知机制，缺点是需要额外的代码实现通知机制；

提供的 API 接口：

（1） 参数初始化：ListMP_Params_init()
（2） 创建/销毁：ListMP_create()/ListMP_delete()
（3） 打开/关闭：ListMP_open()/ListMP_close()
（4） 相关链表操作：

- ✧ 判断链表空：ListMP_empty()
- ✧ 获取保护锁：ListMP_getGate()
- ✧ 获取链表头/表尾：ListMP_getHead()/ListMP_getTail()
- ✧ 链表插入操作：ListMP_insert()

- ✧ 获取链表上游元素/下游元素：ListMP_next()/ListMP_prev()
- ✧ 插入元素到链表头/尾：ListMP_putHead()/ListMP_putTail()
- ✧ 删除元素：ListMP_remove()

8.3.7 ex07_gatemp

示例名字：GateMP

功能说明：GateMP 是针对于多处理器共享资源的一种保护机制，就如其名字一样，把共享资源比作房子，那么 GateMP 就是这个房子的门。GateMP 组件实现了开关门的机制，用于保护共享资源一次只被一个处理器读写。根据 SOC 硬件资源配置的不同，GateMP 的实现有所不同。对于硬件支持 Hardware Spinlock 的可以基于 H/W spinlock 来实现 GateHwSpinlock；而对于没有该硬件资源的系统中，则使用软件方法（Peterson 算法）来实现 GatePeterson。

GateMP 组件框架如下：

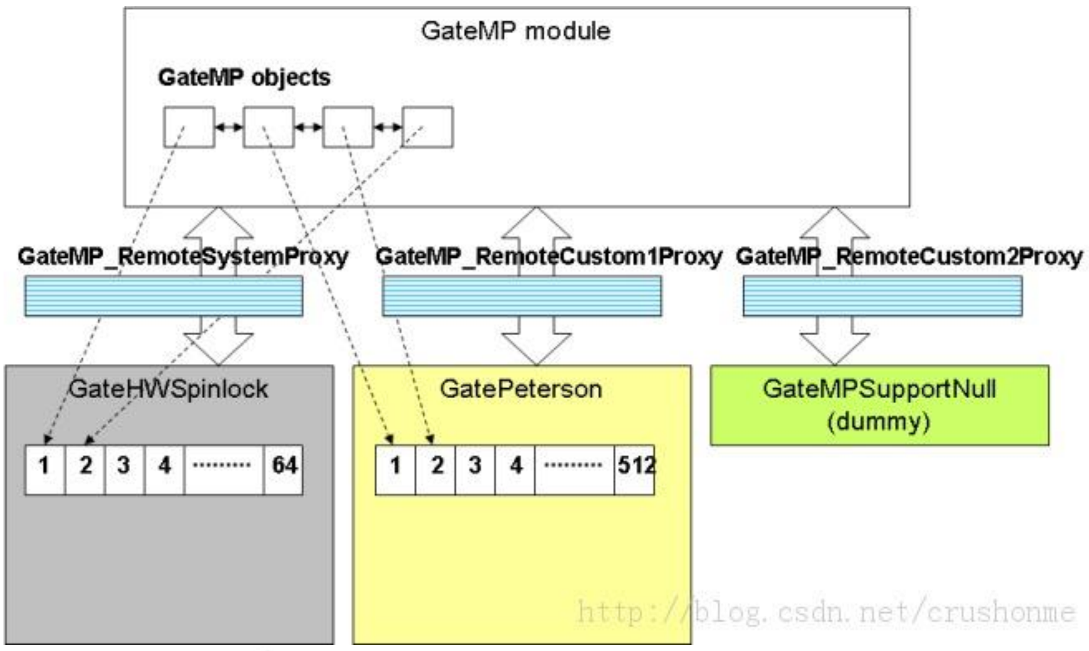
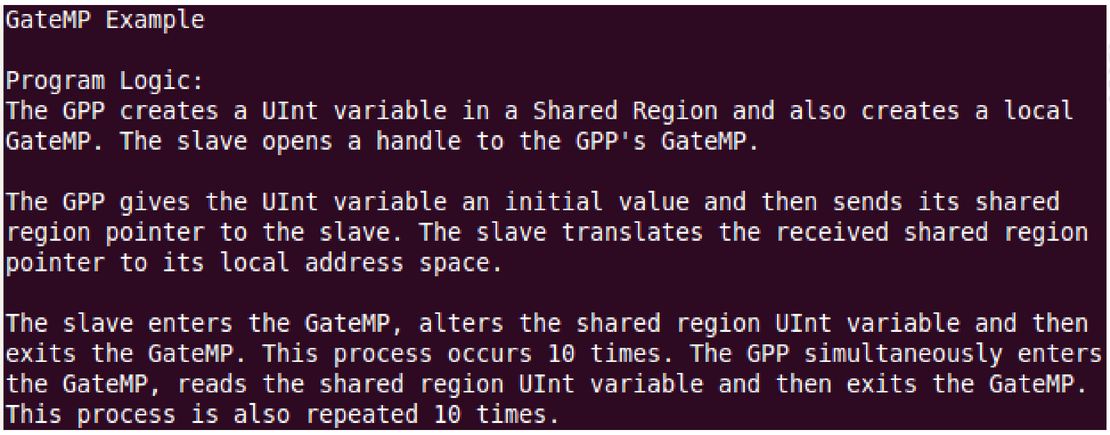


图 61

参考英文资料：



```
GateMP Example

Program Logic:
The GPP creates a UInt variable in a Shared Region and also creates a local
GateMP. The slave opens a handle to the GPP's GateMP.

The GPP gives the UInt variable an initial value and then sends its shared
region pointer to the slave. The slave translates the received shared region
pointer to its local address space.

The slave enters the GateMP, alters the shared region UInt variable and then
exits the GateMP. This process occurs 10 times. The GPP simultaneously enters
the GateMP, reads the shared region UInt variable and then exits the GateMP.
This process is also repeated 10 times.
```

图 62

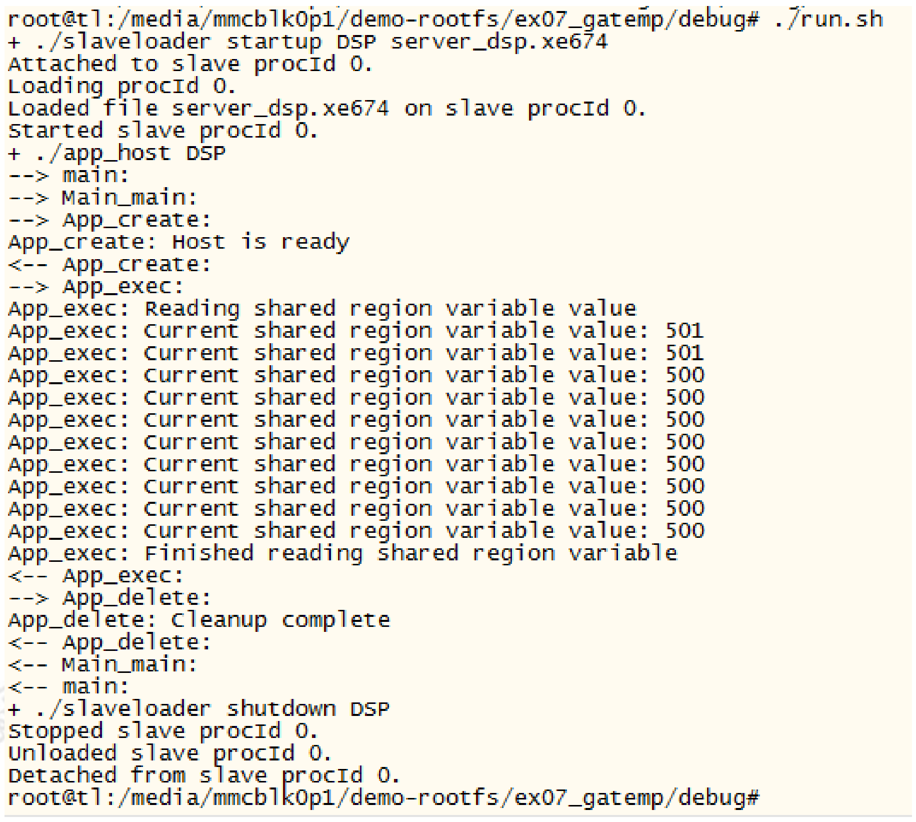
运行命令：

Target#　　cd /ex07_gatemp/debug/

Target#　　ls

Target#　　./run.sh

成功运行提示如下图：



```
root@tl:/media/mmcblk0p1/demo-rootfs/ex07_gatemp/debug# ./run.sh
+ ./slaveloader startup DSP server_dsp.xe674
Attached to slave procId 0.
Loading procId 0.
Loaded file server_dsp.xe674 on slave procId 0.
Started slave procId 0.
+ ./app_host DSP
--> main:
--> Main_main:
--> App_create:
App_create: Host is ready
<-- App_create:
--> App_exec:
App_exec: Reading shared region variable value
App_exec: Current shared region variable value: 501
App_exec: Current shared region variable value: 501
App_exec: Current shared region variable value: 500
App_exec: Current shared region variable value: 500
App_exec: Current shared region variable value: 500
App_exec: Current shared region variable value: 500
App_exec: Current shared region variable value: 500
App_exec: Current shared region variable value: 500
App_exec: Current shared region variable value: 500
App_exec: Current shared region variable value: 500
App_exec: Finished reading shared region variable
<-- App_exec:
--> App_delete:
App_delete: Cleanup complete
<-- App_delete:
<-- Main_main:
<-- main:
+ ./slaveloader shutdown DSP
Stopped slave procId 0.
Unloaded slave procId 0.
Detached from slave procId 0.
root@tl:/media/mmcblk0p1/demo-rootfs/ex07_gatemp/debug#
```

图 63

提供的 API 接口：

（1）　初始化：GateMP_Params_init()

（2）　创建/删除：GateMP_create()/GateMP_delete()

（3）　打开/关闭：GateMP_open()/GateMP_close()

（4）　进入/离开保护：GateMP_enter()/GateMP_leave()

（5）　获取当前的保护类型：GateMP_getLocalProtect()/GateMP_getRemoteProtect()

8.3.8 ex08_ringio

示例名字：RingIO

功能说明：该组件提供基于数据流的循环缓冲区。该组件允许在共享存储空间创建循环缓冲区，不同的处理都能够读取或者写入循环缓冲区。RingIO 组件允许通过写指针来获取数据缓冲区的空存储空间，当该存储空间被释放之后，相应存储空间可以被再次写入。

RingIO 组件允许读指针获取缓冲区中读取空间的有效数据。当被释放之后，相应存储空间的数据被标记为无效。每个 RingIO 实体拥有一个读指针和一个写指针。RingIO 组件也有 API 函数可以使能数据属性的同步传输。如：EOS(End Of Stream)、事件戳、流偏移地址等，也可能伴随循环缓冲区的偏移值。

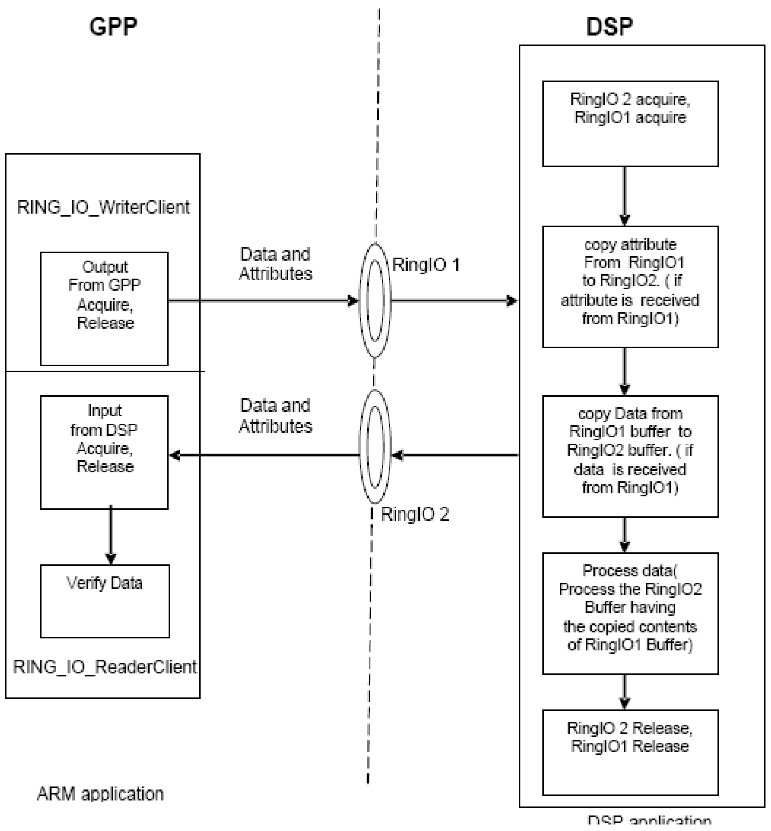
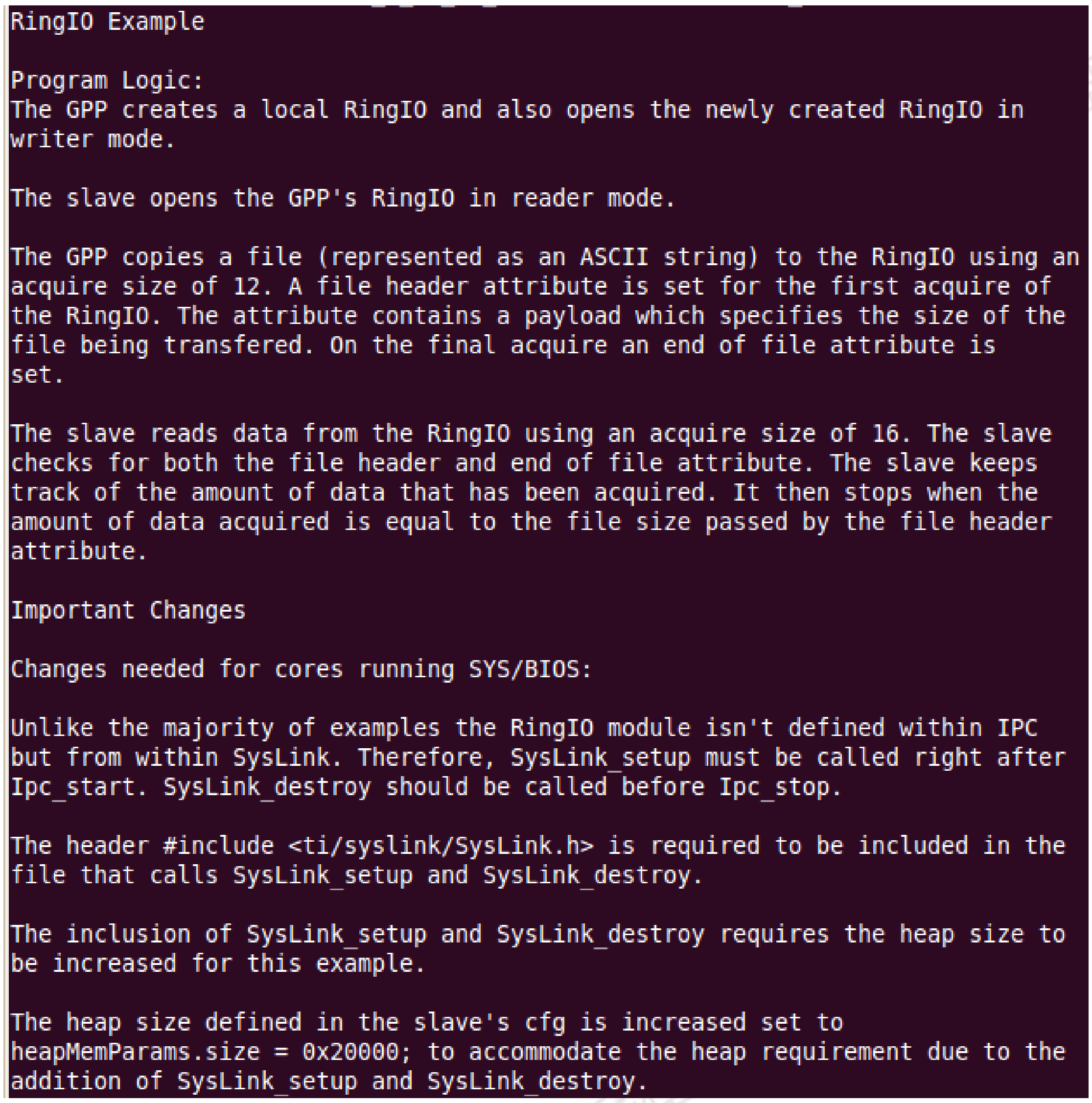

GPP
DSP
RING_IO_WriterClient
Output From GPP Acquire, Release
Data and Attributes
RingIO 1
Input from DSP Acquire, Release
Data and Attributes
RingIO 2
Verify Data
RING_IO_ReaderClient
ARM application
RingIO 2 acquire, RingIO1 acquire
copy attribute From RingIO1 to RingIO2. (if attribute is received from RingIO1)
copy Data from RingIO1 buffer to RingIO2 buffer. (if data is received from RingIO1)
Process data(Process the RingIO2 Buffer having the copied contents of RingIO1 Buffer)
RingIO 2 Release, RingIO1 Release
DSP application

图 64

参考英文资料：



```
RingIO Example

Program Logic:
The GPP creates a local RingIO and also opens the newly created RingIO in
writer mode.

The slave opens the GPP's RingIO in reader mode.

The GPP copies a file (represented as an ASCII string) to the RingIO using an
acquire size of 12. A file header attribute is set for the first acquire of
the RingIO. The attribute contains a payload which specifies the size of the
file being transfered. On the final acquire an end of file attribute is
set.

The slave reads data from the RingIO using an acquire size of 16. The slave
checks for both the file header and end of file attribute. The slave keeps
track of the amount of data that has been acquired. It then stops when the
amount of data acquired is equal to the file size passed by the file header
attribute.

Important Changes

Changes needed for cores running SYS/BIOS:

Unlike the majority of examples the RingIO module isn't defined within IPC
but from within SysLink. Therefore, SysLink_setup must be called right after
Ipc_start. SysLink_destroy should be called before Ipc_stop.

The header #include <ti/syslink/SysLink.h> is required to be included in the
file that calls SysLink_setup and SysLink_destroy.

The inclusion of SysLink_setup and SysLink_destroy requires the heap size to
be increased for this example.

The heap size defined in the slave's cfg is increased set to
heapMemParams.size = 0x20000; to accommodate the heap requirement due to the
addition of SysLink_setup and SysLink_destroy.
```

图 65

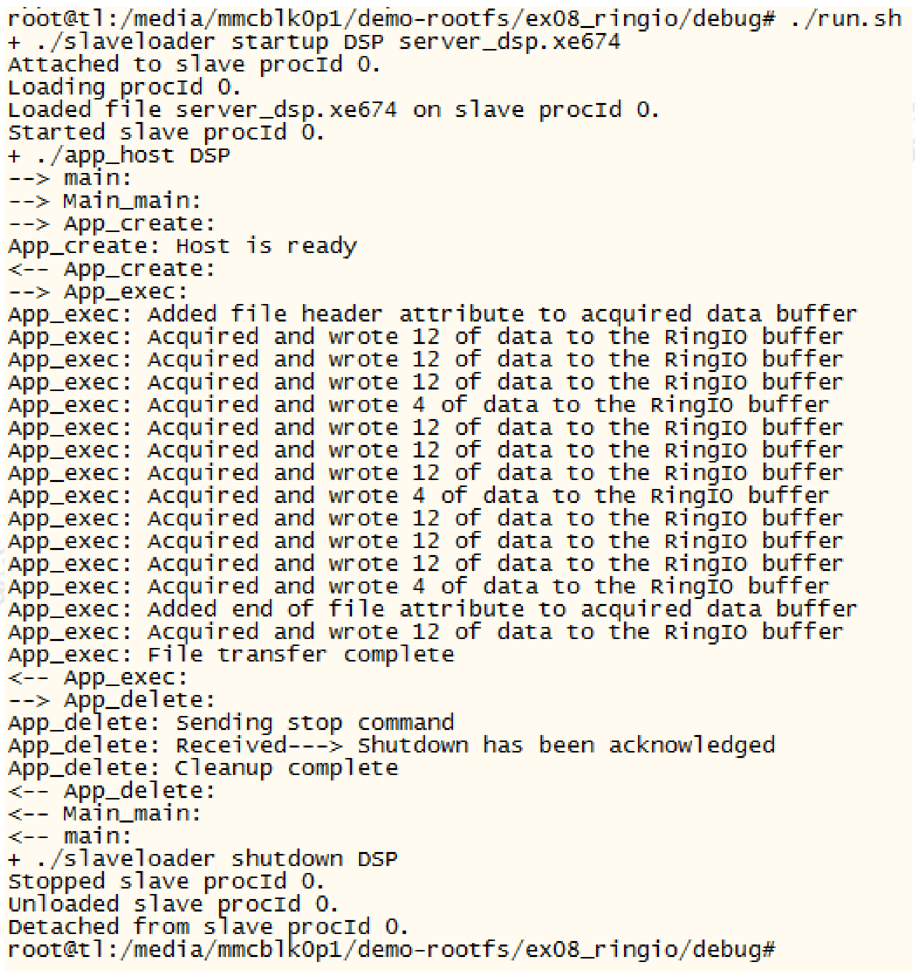
运行命令：

Target# cd /ex08_ringio/debug/

Target# ls

Target# ./run.sh

成功运行提示如下图：



```
root@tl:/media/mmcblk0p1/demo-rootfs/ex08_ringio/debug# ./run.sh
+ ./slaveloader startup DSP server_dsp.xe674
Attached to slave procId 0.
Loading procId 0.
Loaded file server_dsp.xe674 on slave procId 0.
Started slave procId 0.
+ ./app_host DSP
--> main:
--> Main_main:
--> App_create:
App_create: Host is ready
<-- App_create:
--> App_exec:
App_exec: Added file header attribute to acquired data buffer
App_exec: Acquired and wrote 12 of data to the RingIO buffer
App_exec: Acquired and wrote 12 of data to the RingIO buffer
App_exec: Acquired and wrote 12 of data to the RingIO buffer
App_exec: Acquired and wrote 4 of data to the RingIO buffer
App_exec: Acquired and wrote 12 of data to the RingIO buffer
App_exec: Acquired and wrote 12 of data to the RingIO buffer
App_exec: Acquired and wrote 12 of data to the RingIO buffer
App_exec: Acquired and wrote 4 of data to the RingIO buffer
App_exec: Acquired and wrote 12 of data to the RingIO buffer
App_exec: Acquired and wrote 12 of data to the RingIO buffer
App_exec: Acquired and wrote 12 of data to the RingIO buffer
App_exec: Acquired and wrote 4 of data to the RingIO buffer
App_exec: Added end of file attribute to acquired data buffer
App_exec: Acquired and wrote 12 of data to the RingIO buffer
App_exec: File transfer complete
<-- App_exec:
--> App_delete:
App_delete: Sending stop command
App_delete: Received---> Shutdown has been acknowledged
App_delete: Cleanup complete
<-- App_delete:
<-- Main_main:
<-- main:
+ ./slaveloader shutdown DSP
Stopped slave procId 0.
Unloaded slave procId 0.
Detached from slave procId 0.
root@tl:/media/mmcblk0p1/demo-rootfs/ex08_ringio/debug#
```

图 66

特点：

（1） 仅支持一个读者和一个写者；

（2） 读写相对独立，可以在不同的进程或者处理器中同时进行读写操作；

提供的 API 接口：

（1） **RingIO 参数初始化：**RingIO_Params_init()

（2） **创建/删除 RingIO 对象：**RingIO_create()/RingIO_delete()

（3） **打开/关闭 RingIO 对象：**RingIO_open()/RingIO_close()，RingIO_openByAddr()

（4） **获取共享内存请求：**RingIO_sharedMemReq()

（5） **注册/注销 RingIO 通知：**RingIO_registerNotifier()/RingIO_unregisterNotifier()

（6） **强制发送 RingIO 通知：**RingIO_sendNotify()

（7） **获取 RingIO 通知类型**：RingIO_setNotifyType()

（8） **设置/获取水印标志/通知类型**：RIngIO_setWaterMark()/RIngIO_getWaterMark()

（9） **获取/释放 RingIO 数据**：RingIO_acquire()/RingIO_release()

（10） **设置/获取 RingIO 属性**：RingIO_setvAttribute()/RingIO_getvAttribute()

（11） **设置/获取 RingIO 固定大小的属性**：RingIO_setAttribute()/RingIO_getAttribute()

（12） **刷新 RingIO 的 buffer**：RingIO_flush()

（13） **获取有效/空 buffer 大小**：RingIO_getValidSize()/RingIO_getEmptySize()

（14） **获取有效/空属性大小**：RingIO_getValidAttrSize()/RingIO_getEmptyAttrSize()

（15） **获取需求 buffer 的大小/位置**：RingIO_getAcquiredSize()/
RingIO_getAcquiredOffset()

8.3.9 ex09_readwrite

示例名字：ProcMgr read/write

功能说明：ProcMgr read/write 示例阐明了大缓冲区通过直接读写 DSP 内部 RAM 来进行传输的概念。它实现了在 GPP 端和使用 ProcMgr_read()和 ProcMgr_write() API 的 DSP 端以及两个 DSP 端之间的大尺寸数据缓冲器之间的数据与信息的传递和转换。"ProcMgr read/write"示例中数据与信息流向图如下：

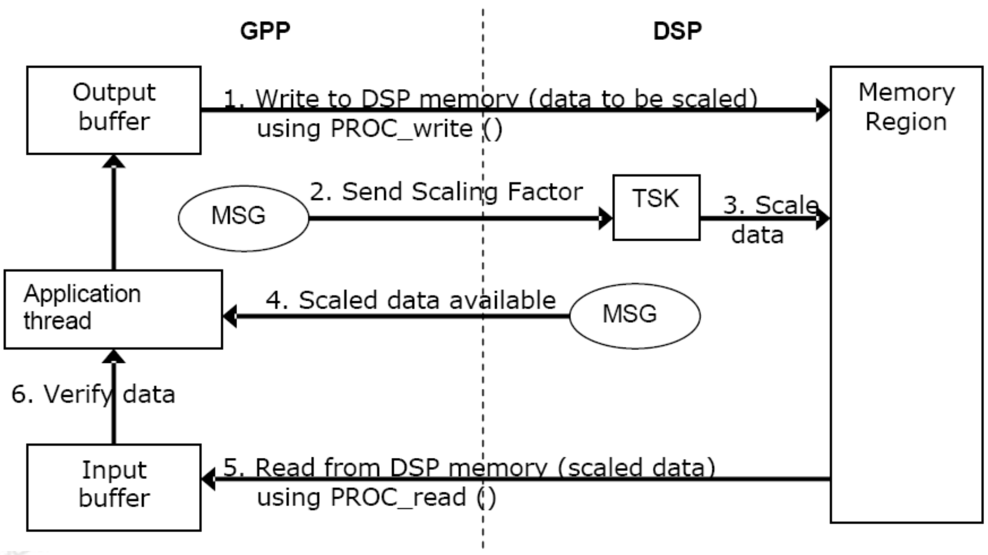


图 67

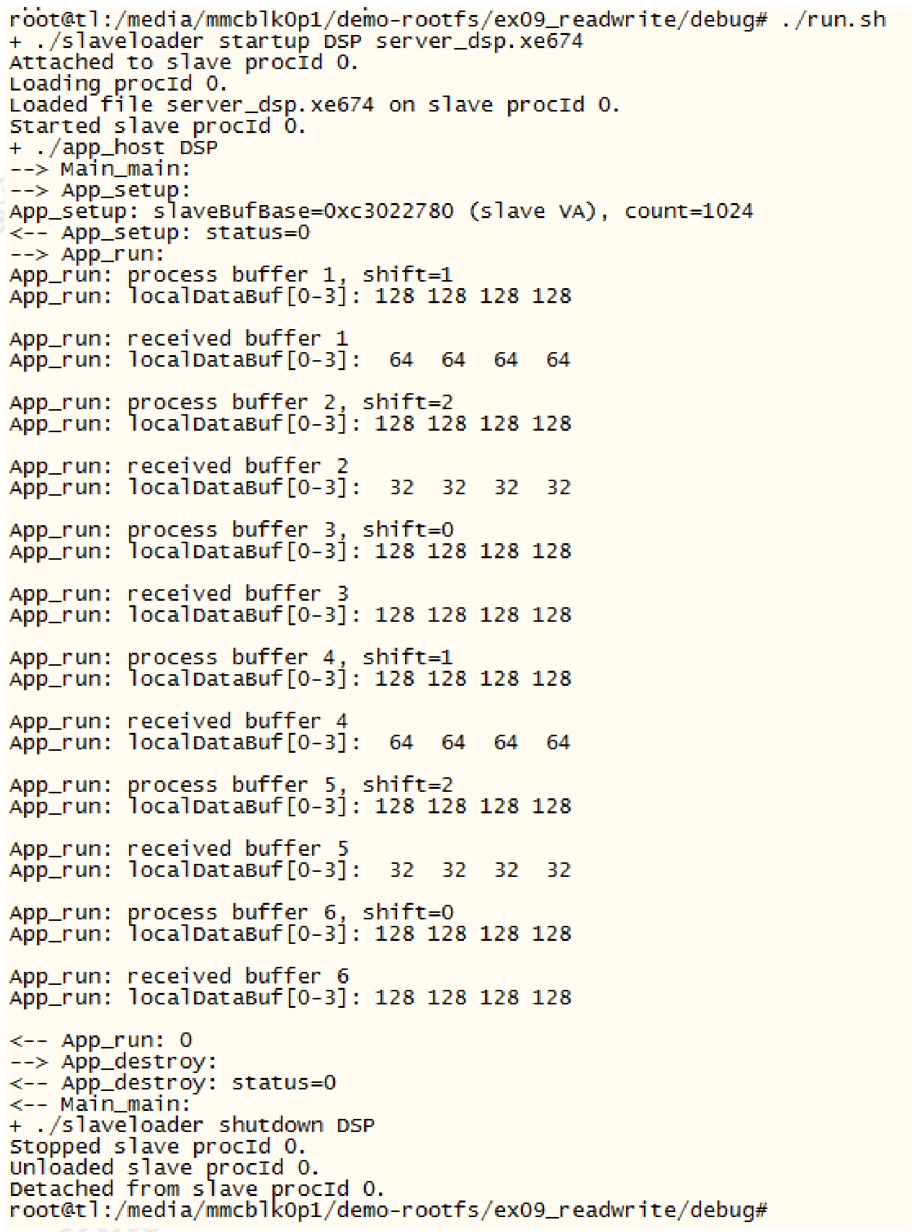
运行命令：

Target# cd /ex09_readwrite/debug/

Target# ls

Target# ./run.sh

成功运行提示如下图：



```
root@tl:/media/mmcblk0p1/demo-rootfs/ex09_readwrite/debug# ./run.sh
+ ./slaveloader startup DSP server_dsp.xe674
Attached to slave procId 0.
Loading procId 0.
Loaded file server_dsp.xe674 on slave procId 0.
Started slave procId 0.
+ ./app_host DSP
--> Main_main:
--> App_setup:
App_setup: slaveBufBase=0xc3022780 (slave VA), count=1024
<-- App_setup: status=0
--> App_run:
App_run: process buffer 1, shift=1
App_run: localDataBuf[0-3]: 128 128 128 128

App_run: received buffer 1
App_run: localDataBuf[0-3]:  64  64  64  64

App_run: process buffer 2, shift=2
App_run: localDataBuf[0-3]: 128 128 128 128

App_run: received buffer 2
App_run: localDataBuf[0-3]:  32  32  32  32

App_run: process buffer 3, shift=0
App_run: localDataBuf[0-3]: 128 128 128 128

App_run: received buffer 3
App_run: localDataBuf[0-3]: 128 128 128 128

App_run: process buffer 4, shift=1
App_run: localDataBuf[0-3]: 128 128 128 128

App_run: received buffer 4
App_run: localDataBuf[0-3]:  64  64  64  64

App_run: process buffer 5, shift=2
App_run: localDataBuf[0-3]: 128 128 128 128

App_run: received buffer 5
App_run: localDataBuf[0-3]:  32  32  32  32

App_run: process buffer 6, shift=0
App_run: localDataBuf[0-3]: 128 128 128 128

App_run: received buffer 6
App_run: localDataBuf[0-3]: 128 128 128 128

<-- App_run: 0
--> App_destroy:
<-- App_destroy: status=0
<-- Main_main:
+ ./slaveloader shutdown DSP
Stopped slave procId 0.
Unloaded slave procId 0.
Detached from slave procId 0.
root@tl:/media/mmcblk0p1/demo-rootfs/ex09_readwrite/debug#
```

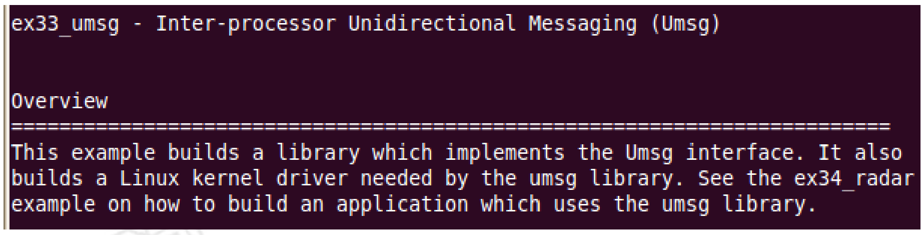
图 68

8.3.10 ex33_umsg

示例名字：umsg(Inter-processor Unidirectional Messaging)

功能说明：编译 usmg 库文件和 umsg 相关驱动。

参考英文资料：



```
ex33_umsg - Inter-processor Unidirectional Messaging (Umsg)


Overview
=========================================================================
This example builds a library which implements the Umsg interface. It also
builds a Linux kernel driver needed by the umsg library. See the ex34_radar
example on how to build an application which uses the umsg library.
```

图 69

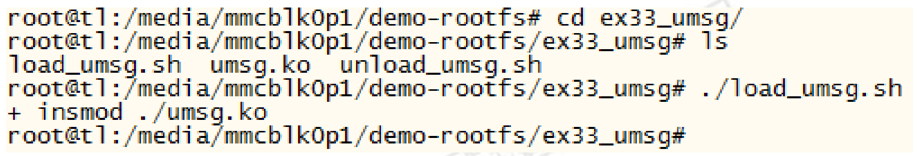
运行命令：

Target# cd /ex33_umsg/

Target# ls

Target# ./load_umsg.sh

成功运行提示如下图：



```
root@tl:/media/mmcblk0p1/demo-rootfs# cd ex33_umsg/
root@tl:/media/mmcblk0p1/demo-rootfs/ex33_umsg# ls
load_umsg.sh  umsg.ko  unload_umsg.sh
root@tl:/media/mmcblk0p1/demo-rootfs/ex33_umsg# ./load_umsg.sh
+ insmod ./umsg.ko
root@tl:/media/mmcblk0p1/demo-rootfs/ex33_umsg#
```

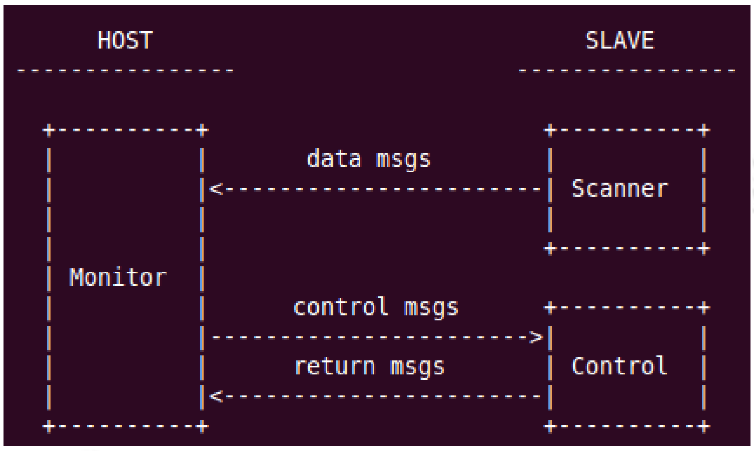
图 70

备注：由于 ex34_radar 示例用到了 ex33_umsg 示例编译出来的驱动程序和库文件，因此在使用 ex34_radar 示例之前，一定要运行 ex33_umsg 示例。

8.3.11 ex34_radar

示例名字：radar

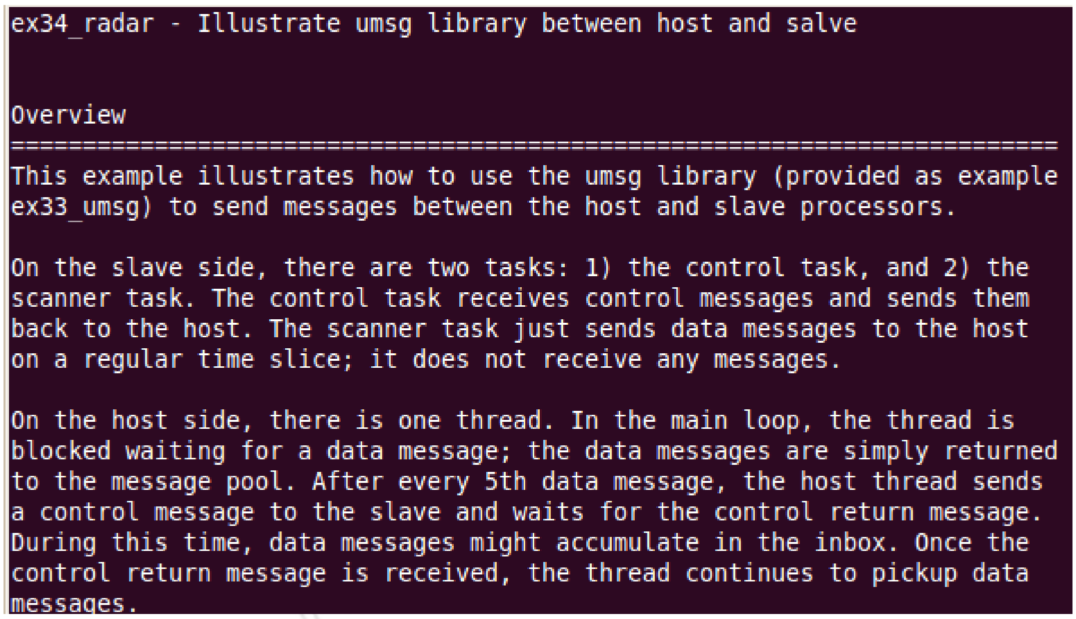
功能说明：阐明如何利用 umsg 库在 GPP 和 DSP 之间传递信息。



```
     HOST                                          SLAVE
-----------------                            -----------------

  +----------+                                +----------+
  |          |          data msgs             |          |
  |          |<-------------------------------| Scanner  |
  |          |                                |          |
  |          |                                +----------+
  | Monitor  |
  |          |         control msgs           +----------+
  |          |------------------------------->|          |
  |          |         return msgs            | Control  |
  |          |<-------------------------------|          |
  +----------+                                +----------+
```

图 71

参考英文资料：



```
ex34_radar - Illustrate umsg library between host and salve


Overview
=========================================================================
This example illustrates how to use the umsg library (provided as example
ex33_umsg) to send messages between the host and slave processors.

On the slave side, there are two tasks: 1) the control task, and 2) the
scanner task. The control task receives control messages and sends them
back to the host. The scanner task just sends data messages to the host
on a regular time slice; it does not receive any messages.

On the host side, there is one thread. In the main loop, the thread is
blocked waiting for a data message; the data messages are simply returned
to the message pool. After every 5th data message, the host thread sends
a control message to the slave and waits for the control return message.
During this time, data messages might accumulate in the inbox. Once the
control return message is received, the thread continues to pickup data
messages.
```

图 72

Target#　　cd /ex34_radar/debug/

Target#　　ls

Target#　　./run.sh

成功运行提示如下图：

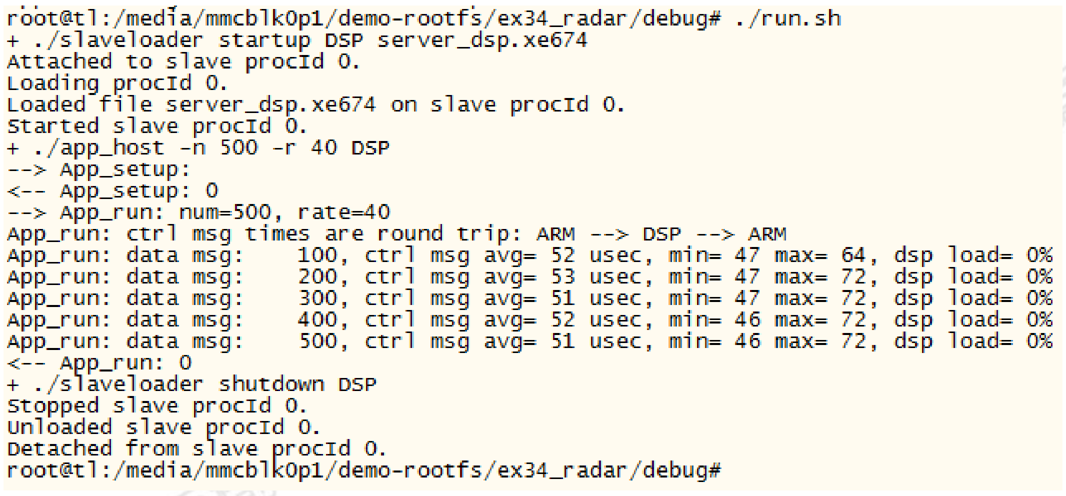

```
root@tl:/media/mmcblk0p1/demo-rootfs/ex34_radar/debug# ./run.sh
+ ./slaveloader startup DSP server_dsp.xe674
Attached to slave procId 0.
Loading procId 0.
Loaded file server_dsp.xe674 on slave procId 0.
Started slave procId 0.
+ ./app_host -n 500 -r 40 DSP
--> App_setup:
<-- App_setup: 0
--> App_run: num=500, rate=40
App_run: ctrl msg times are round trip: ARM --> DSP --> ARM
App_run: data msg:    100, ctrl msg avg= 52 usec, min= 47 max= 64, dsp load= 0%
App_run: data msg:    200, ctrl msg avg= 53 usec, min= 47 max= 72, dsp load= 0%
App_run: data msg:    300, ctrl msg avg= 51 usec, min= 47 max= 72, dsp load= 0%
App_run: data msg:    400, ctrl msg avg= 52 usec, min= 46 max= 72, dsp load= 0%
App_run: data msg:    500, ctrl msg avg= 51 usec, min= 46 max= 72, dsp load= 0%
<-- App_run: 0
+ ./slaveloader shutdown DSP
Stopped slave procId 0.
Unloaded slave procId 0.
Detached from slave procId 0.
root@tl:/media/mmcblk0p1/demo-rootfs/ex34_radar/debug#
```

图 73

更多帮助

销售邮箱：sales@tronlong.com

技术邮箱：support@tronlong.com

创龙总机：020-8998-6280

技术热线：020-3893-9734

创龙官网：www.tronlong.com

技术论坛：www.51ele.net

线上商城：https://tronlong.taobao.com

TMS320C6748、OMAPL138 交流群：227961486、324023586

TI 中文论坛：http://www.deyisupport.com/

TI 英文论坛：http://e2e.ti.com/

TI 官网：www.ti.com

TI WIKI：http://processors.wiki.ti.com/