|  |
| --- |
|  |
| Keystone II guide on running IPC examples |
|  |

July 2016

Contents

[Prerequisites 1](#_Toc456359403)

[Hardware 1](#_Toc456359404)

[Software 1](#_Toc456359405)

[1. Updating the U-BOOT 3](#_Toc456359406)

[Update SPI NOR Flash with U-boot GPH image 3](#_Toc456359407)

[2. Set the BOOT switch settings as SPI boot mode and boot the U-boot 4](#_Toc456359408)

[3. Flash UBIFS into NAND and boot Linux 5](#_Toc456359409)

[4. Build IPC package 10](#_Toc456359410)

[Lab 1: How to build and run ex44\_compute on target 12](#_Toc456359411)

[Build the ex44\_compute example in a host Ubuntu machine. 12](#_Toc456359412)

[Copy the ex44\_compute into the target filesystem. 13](#_Toc456359413)

[Execute the ex44\_compute example on the targer EVM. 14](#_Toc456359414)

[Output Logs 14](#_Toc456359415)

[Lab 2: How to build and run ex02\_MessageQ on target 19](#_Toc456359416)

[Build the ex02\_ messageq example in a host Ubuntu machine. 19](#_Toc456359417)

[Copy the ex02\_messageq into the target file system. 20](#_Toc456359418)

[Execute the ex02\_messageq example on the target EVM. 20](#_Toc456359419)

[Output Logs- K2E EVM 21](#_Toc456359420)

[Output Logs- K2H EVM 21](#_Toc456359421)

[Lab 3: How to build and run ex45\_host on target 22](#_Toc456359422)

[Build the ex45\_host example in a host Ubuntu machine. 22](#_Toc456359423)

[Copy the ex45\_host into the target file system. 24](#_Toc456359424)

[Execute the ex45\_host example on the target EVM. 24](#_Toc456359425)

[Output Logs 25](#_Toc456359426)

[Lab 4: How to build and run ex46\_graph on target 26](#_Toc456359427)

[Build the ex46\_graph example in a host Ubuntu machine. 26](#_Toc456359428)

[Copy the ex46\_graph into the target file system. 28](#_Toc456359429)

[Execute the ex46\_graph example on the target EVM 28](#_Toc456359430)

[Output Logs 31](#_Toc456359431)

[Debugging steps for DSP side programs for the IPC examples. 34](#_Toc456359432)

# Prerequisites

The following hardware and software are needed to perform the labs in this manual.

## Hardware

Customer is expected to have any one of the EVMS and Ubuntu 14.04 64-bit.

1. EVM k2H Revision 3.0 and above.

<http://processors.wiki.ti.com/index.php/EVMK2H_Hardware_Setup>

1. EVM k2E Revision 1.0.2.2 and above.

<http://processors.wiki.ti.com/index.php/EVMK2E_Hardware_Setup>

## Software

The following software packages must be pre-installed on the Customer’s PC. It is assumed that the customer’s PC is attached to the local network and has access to internet.

1. Download the Processor SDK 2.0.2

PROCESSOR-SDK-LINUX-K2E: <http://www.ti.com/tool/PROCESSOR-SDK-K2H>

PROCESSOR-SDK-LINUX-K2HK: <http://www.ti.com/tool/PROCESSOR-SDK-K2E>

1. Install the processor SDK 2.0.2

As per the installation instructions given in the Install guide of the package, install the Processor SDK 2.0.2 on Linux machines.

This will install the IPC packages as well.

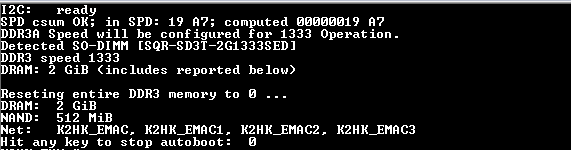
## Updating the U-BOOT

The U-BOOT shall be flashed into SPI / NAND memories.

### Update SPI NOR Flash with U-boot GPH image

The following process is used to update the U-BOOT image in SPI Flash. It must be done every time a new release of PSDK is used.

1. Power cycle the EVM and stop the autoboot by pressing any key.



1. The image sub-directory of the processor SDK release has a .gph file - **u-boot-spi-k2hk-evm.gph.** This file shall be copied to the TFTP root directory
2. Make sure the tftp server is running. Then issue the following commands to U-Boot console:

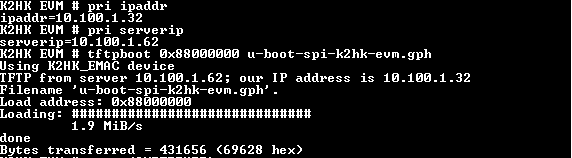
**dhcp**



**setenv ipaddr 10.100.1.32**

**setenv serverip 10.100.1.62**

**tftpboot 0x88000000 u-boot-spi-k2hk-evm.gph**

****

**sf probe**

**(Offset is depends on u-boot image size)**

**sf erase 0 0x100000**

**sf write 0x88000000 0 0x100000**

****

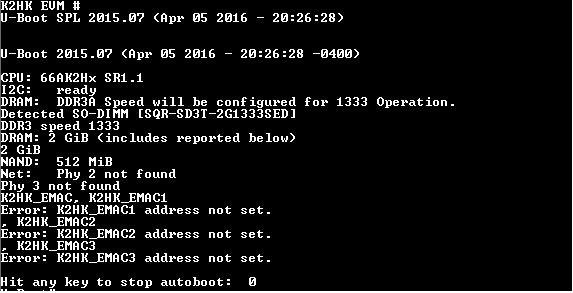
## Set the BOOT switch settings as SPI boot mode and boot the U-boot

Settings DIP switch to SPI boot mode.

K2H EVM - SW1 and K2E EVM – SW1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Pin# | 1 | 2 | 3 | 4 |
| Position | OFF | OFF | ON | OFF |

Do a power on reset and observe that the newly update u-boot is up and running



## Flash UBIFS into NAND and boot Linux

Copy the ubifs image into the RAM address, 0x82000000

**tftpboot 0x82000000 tisdk-server-rootfs-image-k2hk-evm.ubi**

**nand erase.part ubifs**

**pri burn\_ubi**

**( check all the params are right )**

**run burn\_ubi**

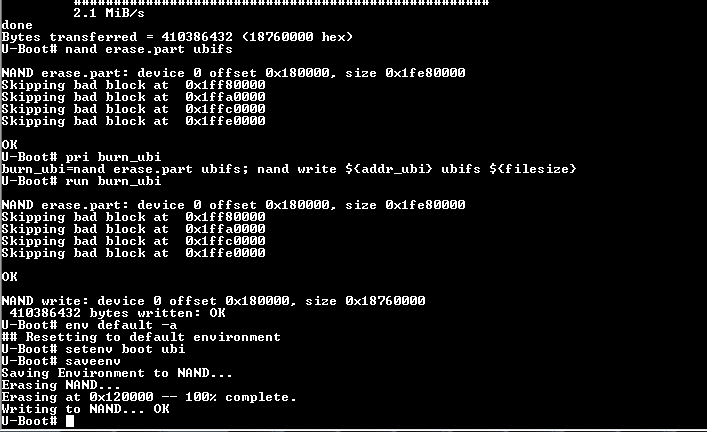
**env default -a**

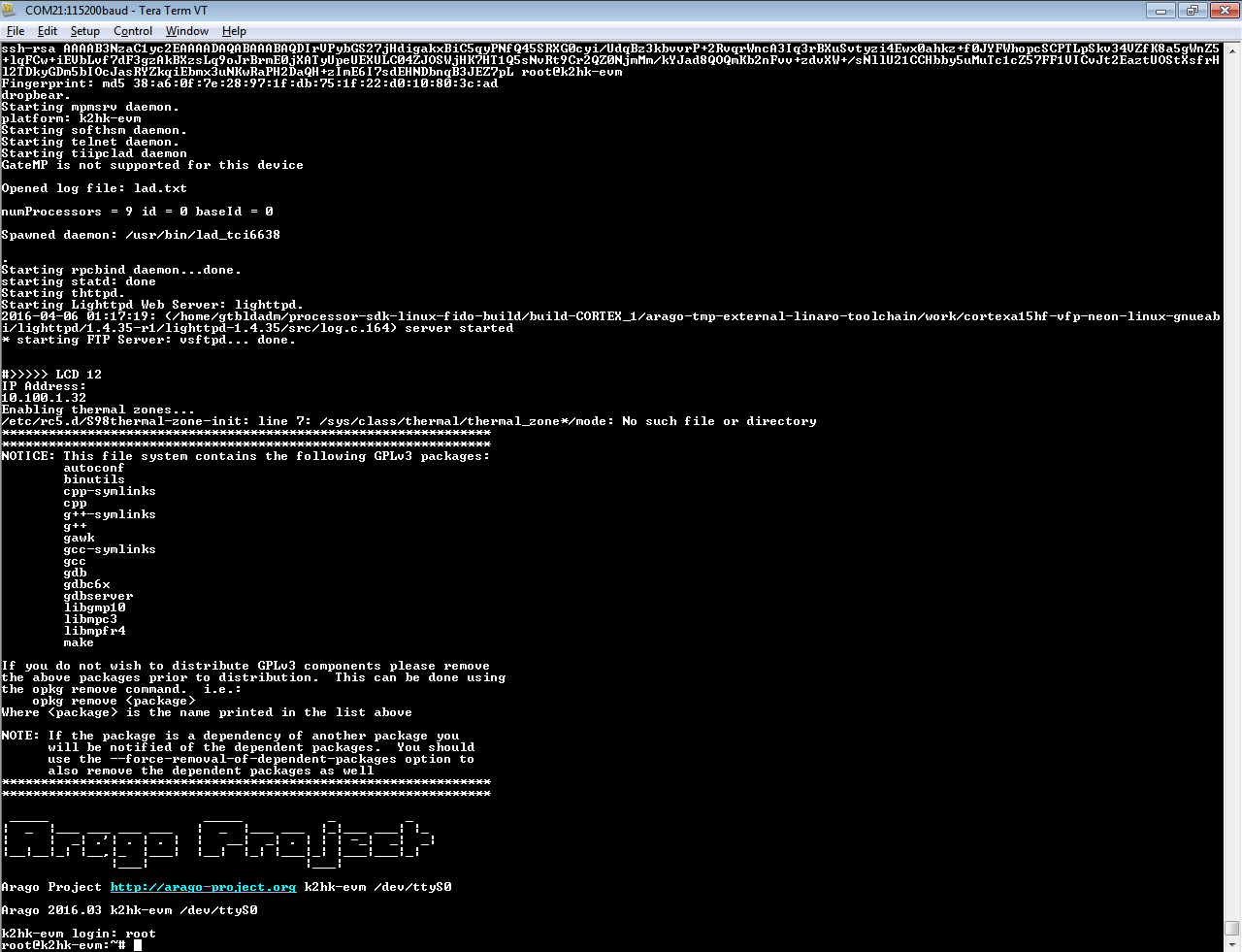
**setenv boot ubi**

**saveenv**

**boot**







1. When booting ends, login as **root** (no password)

## Build IPC package

Along with the Processor SDK package, IPC package will also get installed.

For example, if you install the PSDK version, “ ti-processor-sdk-rtos-k2hk-evm-02.00.02.11-Linux-x86-install.bin”, the IPC version, “ ipc\_3\_42\_00\_02”

1. Go to the path where the IPC is installed.

**cd /opt/ti/ipc\_3\_42\_00\_02**

1. Create a new directory to store the IPC libraries :

**mkdir IPC\_install**

1. Modify the products.mak file appropriate to your host machine.

DEPOT = /opt/ti

# Platform to build for

# supported platforms (choose one):

# OMAPL138, OMAP54XX, DRA7XX, 66AK2G, 66AK2E, TCI6630, TCI6636, TCI6638,

# TDA3XX

#

# Note, this is used for Linux, QNX and BIOS builds

#

PLATFORM = 66AK2E

# Destination when installing the built binaries

#

# Note, this is used for Linux (if you use ipc-linux.mak to run the

# configure command), QNX and BIOS.

#

DESTDIR = /opt/ti/ipc\_3\_42\_00\_02/IPC\_install

#################### IPC Linux ####################

# Set up required cross compiler path for IPC Linux configuration and build

#

TOOLCHAIN\_LONGNAME = arm-linux-gnueabihf

TOOLCHAIN\_INSTALL\_DIR = /home/shankari/workdir/keystone/processor\_sdk\_02\_00\_02\_11/gcc-linaro-4.9-2015.05-x86\_64\_arm-linux-gnueabihf

TOOLCHAIN\_PREFIX = $(TOOLCHAIN\_INSTALL\_DIR)/bin/$(TOOLCHAIN\_LONGNAME)-

# Path to Linux Kernel - needed to build the IPC user libraries

#

KERNEL\_INSTALL\_DIR = /home/shankari/ti/ti-processor-sdk-linux-k2hk-evm-02.00.02.11/board-support/linux-4.1.18+gitAUTOINC+bbe8cfc1da-gbbe8cfc

# Optional: Specify the Address Family for RPMSG. This value is specified

# either from the Linux kernel specified by KERNEL\_INSTALL\_DIR above, or

# the make variable AF\_RPMSG below. Do not use both.

#

AF\_RPMSG =

# Optional: Path to DRM Library

#

DRM\_PREFIX =

# Optional: Path to TI Linux Utils product

#

CMEM\_INSTALL\_DIR =

#################### IPC QNX ####################

# Path to QNX tools installation

#

QNX\_INSTALL\_DIR =

# Optional: Any additional compile options

#

QNX\_CFLAGS =

#################### IPC Bios ####################

# Path to required dependencies for IPC BIOS builds

#

XDC\_INSTALL\_DIR = $(DEPOT)/xdctools\_3\_32\_00\_06\_core

BIOS\_INSTALL\_DIR = $(DEPOT)/bios\_6\_45\_01\_29

# Do you want to build SMP-enabled libraries (if supported for your target)?

# Set to either 0 (disabled) or 1 (enabled)

#

BIOS\_SMPENABLED=1

# Path to various cgtools

#

ti.targets.elf.C64P =

ti.targets.elf.C64P\_big\_endian =

ti.targets.elf.C64T =

ti.targets.elf.C66 = /opt/ti/ccsv6/tools/compiler/ti-cgt-c6000\_8.1.0

ti.targets.elf.C66\_big\_endian =

ti.targets.elf.C674 =

ti.targets.arm.elf.Arm9 =

ti.targets.arm.elf.A8F =

ti.targets.arm.elf.A8Fnv =

ti.targets.arm.elf.M3 =

ti.targets.arm.elf.M4 =

ti.targets.arm.elf.M4F =

ti.targets.arp32.elf.ARP32 =

ti.targets.arp32.elf.ARP32\_far =

gnu.targets.arm.A8F =

gnu.targets.arm.A15F =

1. Build IPC

**cd /opt/ti/ipc\_3\_42\_00\_02**

**make distclean**

**make -f ipc-linux.mak config**

**make**

**make install** --- > This will install the IPC libraries in the folder given in the products.mak

# Lab 1: How to build and run ex44\_compute on target

### Build the ex44\_compute example in a host Ubuntu machine.

1. Create a work folder on your file system.

mkdir work

1. Extract the example located inside the IPC package.

cd work

unzip ex44\_compute.zip

1. Setup the build environment. Edit products.mak and set the install paths as defined by your physical development area. Each example has its own products.mak file; you may also create a products.mak file in the parent directory which will be used by all examples.

edit ex44\_compute/products.mak

# look for other products.mak file to override local settings

ifneq (,$(wildcard $(EXBASE)/../products.mak))

include $(EXBASE)/../products.mak

else

ifneq (,$(wildcard $(EXBASE)/../../products.mak))

include $(EXBASE)/../../products.mak/

# Define IPC\_INSTALL\_DIR since not defined in IPC top-level products.mak

IPC\_INSTALL\_DIR = $(word 1,$(subst /examples, examples,$(CURDIR)))

endif

endif

# By default, the necessary build variables are found/assigned via

# ../products.mak or ../../products.mak, included above. If you want to

# override these variables, or are building this example without

# ../products.mak or ../../products.mak, uncomment and assign the variables

# below.

DEPOT = /opt/ti

#### Linux toolchain ####

TOOLCHAIN\_LONGNAME = arm-linux-gnueabihf

TOOLCHAIN\_INSTALL\_DIR = /home/shankari/workdir/keystone/processor\_sdk\_02\_00\_02\_11/gcc-linaro-4.9-2015.05-x86\_64\_arm-linux-gnueabihf

TOOLCHAIN\_PREFIX = $(TOOLCHAIN\_INSTALL\_DIR)/bin/$(TOOLCHAIN\_LONGNAME)-

#### BIOS-side dependencies ####

BIOS\_INSTALL\_DIR = $(DEPOT)/bios\_6\_45\_01\_29

IPC\_INSTALL\_DIR = $(DEPOT)/ipc\_3\_42\_00\_02

XDC\_INSTALL\_DIR = $(DEPOT)/xdctools\_3\_32\_00\_06\_core

DESTDIR = $(DEPOT)/ipc\_3\_42\_00\_02/IPC\_install

#### BIOS-side toolchains ####

ti.targets.elf.C66 = $(DEPOT)/ccsv6/tools/compiler/ti-cgt-c6000\_8.1.0

# Use this goal to print your product variables.

.show:

@echo "TOOLCHAIN\_LONGNAME = $(TOOLCHAIN\_LONGNAME)"

@echo "TOOLCHAIN\_INSTALL\_DIR = $(TOOLCHAIN\_INSTALL\_DIR)"

@echo "TOOLCHAIN\_PREFIX = $(TOOLCHAIN\_PREFIX)"

@echo "BIOS\_INSTALL\_DIR = $(BIOS\_INSTALL\_DIR)"

@echo "IPC\_INSTALL\_DIR = $(IPC\_INSTALL\_DIR)"

@echo "XDC\_INSTALL\_DIR = $(XDC\_INSTALL\_DIR)"

@echo "DESTDIR = $(DESTDIR)"

@echo "ti.targets.elf.C66 = $(ti.targets.elf.C66)"

1. Build the example. This will build only debug versions of the executables. Edit the lower makefiles and uncomment the release goals to build both debug and release executables.

cd ex44\_compute

make

1. Issue the following commands to clean your example.

cd ex44\_compute

make clean

### Copy the ex44\_compute into the target filesystem.

1. Copy the HOST executable, the DSP executable, and the supporting scripts to your target file system.

ex44\_compute/host/bin/debug/app\_host

ex44\_compute/dsp/bin/debug/compute\_dspN.xe66

ex44\_compute/scripts/patchExec.pl

ex44\_compute/scripts/run\_dsp.sh

ex44\_compute/scripts/run\_host.sh

ex44\_compute/scripts/run\_lad.sh

ex44\_compute/scripts/run\_patch.sh

ex44\_compute/scripts/stop\_dsp.sh

1. Generate the compute binary according to the device ID:

perl patchExec.pl 0 compute\_dspN.xe66 compute\_dspN\_patched.xe66

You can also use the helper script provided with the example.

run\_patch.sh 0

This will generate a new patched DSP executable.

compute\_dspN\_patched.xe66

1. Rebuild your LAD executable. Copy the LAD executable to your target file system.

<IPC Install>/linux/src/daemon/lad\_tci6638

1. Copy to IPC\_INSTALL\_DIR folder your filesystem.

### Execute the ex44\_compute example on the targer EVM.

1. Go to IPC\_install/bin and launch the Lad daemon.

**cd IPC\_install/bin**

**./lad\_tci6638 -r 8 -n 9 -b 0 -l log.txt**

( where –n = number of processors, -b = baseid of the cluster, -r = number of slave side entries )

2. Goto /usr/bin/scripts/

**run run\_dsp.sh**

3. Goto host folder

**./app\_host**

### Output Logs

root@k2hk-evm:/home/ex44\_compute/host/bin/debug# ./app\_host

ResMgr\_threadFxn: -->

ResMgr\_setup: -->

App\_threadFxn: -->

App\_setup: -->

ResMgr\_setup: initalizing IPC

App\_setup: initalizing IPC

ResMgr\_setup: IPC ready, status=0

App\_setup: IPC ready, status=1

ResMgr\_setup: <-- status=0

ResMgr\_exec: -->

ResMgr\_exec: waiting for message

App\_setup: cluster baseId=0, cluster members:

App\_setup: 0 HOST

App\_setup: 1 CORE0

App\_setup: 2 CORE1

App\_setup: 3 CORE2

App\_setup: 4 CORE3

App\_setup: 5 CORE4

App\_setup: 6 CORE5

App\_setup: 7 CORE6

App\_setup: 8 CORE7

App\_setup: MessageQ\_open(PEB\_Proc1)

App\_setup: PEB queue: proc=CORE0, qid=0x00010089

App\_setup: MessageQ\_open(Control\_Proc1)

App\_setup: Control queue: proc=CORE0, qid=0x00010088

App\_setup: MessageQ\_open(PEB\_Proc2)

App\_setup: PEB queue: proc=CORE1, qid=0x00020089

App\_setup: MessageQ\_open(Control\_Proc2)

App\_setup: Control queue: proc=CORE1, qid=0x00020088

App\_setup: MessageQ\_open(PEB\_Proc3)

App\_setup: PEB queue: proc=CORE2, qid=0x00030089

App\_setup: MessageQ\_open(Control\_Proc3)

App\_setup: Control queue: proc=CORE2, qid=0x00030088

App\_setup: MessageQ\_open(PEB\_Proc4)

App\_setup: PEB queue: proc=CORE3, qid=0x00040089

App\_setup: MessageQ\_open(Control\_Proc4)

App\_setup: Control queue: proc=CORE3, qid=0x00040088

App\_setup: MessageQ\_open(PEB\_Proc5)

App\_setup: PEB queue: proc=CORE4, qid=0x00050089

App\_setup: MessageQ\_open(Control\_Proc5)

App\_setup: Control queue: proc=CORE4, qid=0x00050088

App\_setup: MessageQ\_open(PEB\_Proc6)

App\_setup: PEB queue: proc=CORE5, qid=0x00060089

App\_setup: MessageQ\_open(Control\_Proc6)

App\_setup: Control queue: proc=CORE5, qid=0x00060088

App\_setup: MessageQ\_open(PEB\_Proc7)

App\_setup: PEB queue: proc=CORE6, qid=0x00070089

App\_setup: MessageQ\_open(Control\_Proc7)

App\_setup: Control queue: proc=CORE6, qid=0x00070088

App\_setup: MessageQ\_open(PEB\_Proc8)

App\_setup: PEB queue: proc=CORE7, qid=0x00080089

App\_setup: MessageQ\_open(Control\_Proc8)

App\_setup: Control queue: proc=CORE7, qid=0x00080088

App\_setup: start message sent: procId=1 qid=0x00010088

ResMgr\_exec: message received, cmd=1

ResMgr\_exec: cmd=REQUEST, sender=1

ResMgr\_exec: message sent: qid=0x00010088

ResMgr\_exec: waiting for message

App\_setup: start message sent: procId=2 qid=0x00020088

ResMgr\_exec: message received, cmd=1

ResMgr\_exec: cmd=REQUEST, sender=2

ResMgr\_exec: message sent: qid=0x00020088

ResMgr\_exec: waiting for message

App\_setup: start message sent: procId=3 qid=0x00030088

ResMgr\_exec: message received, cmd=1

ResMgr\_exec: cmd=REQUEST, sender=3

ResMgr\_exec: message sent: qid=0x00030088

ResMgr\_exec: waiting for message

App\_setup: start message sent: procId=4 qid=0x00040088

ResMgr\_exec: message received, cmd=1

ResMgr\_exec: cmd=REQUEST, sender=4

ResMgr\_exec: message sent: qid=0x00040088

ResMgr\_exec: waiting for message

App\_setup: start message sent: procId=5 qid=0x00050088

ResMgr\_exec: message received, cmd=1

ResMgr\_exec: cmd=REQUEST, sender=5

ResMgr\_exec: message sent: qid=0x00050088

ResMgr\_exec: waiting for message

App\_setup: start message sent: procId=6 qid=0x00060088

ResMgr\_exec: message received, cmd=1

ResMgr\_exec: cmd=REQUEST, sender=6

ResMgr\_exec: message sent: qid=0x00060088

ResMgr\_exec: waiting for message

App\_setup: start message sent: procId=7 qid=0x00070088

ResMgr\_exec: message received, cmd=1

ResMgr\_exec: cmd=REQUEST, sender=7

ResMgr\_exec: message sent: qid=0x00070088

ResMgr\_exec: waiting for message

App\_setup: start message sent: procId=8 qid=0x00080088

ResMgr\_exec: message received, cmd=1

ResMgr\_exec: cmd=REQUEST, sender=8

ResMgr\_exec: message sent: qid=0x00080088

ResMgr\_exec: waiting for message

App\_setup: TransportQMSS instance created

App\_setup: <-- status=0

App\_exec: -->

App\_exec: message sent to PEB queue: qid=0x00010089

App\_exec: message sent to PEB queue: qid=0x00020089

App\_exec: message sent to PEB queue: qid=0x00030089

App\_exec: message sent to PEB queue: qid=0x00040089

App\_exec: message sent to PEB queue: qid=0x00050089

App\_exec: message sent to PEB queue: qid=0x00060089

App\_exec: message sent to PEB queue: qid=0x00070089

App\_exec: message sent to PEB queue: qid=0x00080089

App\_exec: message sent to Compute queue: qid=0x00010082

App\_exec: message sent to Compute queue: qid=0x00020082

App\_exec: message sent to Compute queue: qid=0x00030082

App\_exec: message sent to Compute queue: qid=0x00040082

App\_exec: message sent to Compute queue: qid=0x00050082

App\_exec: message sent to Compute queue: qid=0x00060082

App\_exec: message sent to Compute queue: qid=0x00070082

App\_exec: message sent to Compute queue: qid=0x00080082

App\_exec: <-- status=0

App\_destroy: -->

App\_destroy: <--

App\_threadFxn: <-- status=0

main: application thread as terminated

ResMgr\_exec: <-- status=0

ResMgr\_destroy: -->

ResMgr\_destroy: <--

ResMgr\_threadFxn: <-- status=0

main: appStatus=0

main: rmStatus=0

main: <-- status=0

=====================================================================

# Lab 2: How to build and run ex02\_MessageQ on target

### Build the ex02\_ messageq example in a host Ubuntu machine.

1. Go to the ex02\_messageq directory inside the IPC package

**cd ex02\_ messageq**

1. Modify the products.mak file and save.

# look for other products.mak file to override local settings

ifneq (,$(wildcard $(EXBASE)/../products.mak))

include $(EXBASE)/../products.mak

else

ifneq (,$(wildcard $(EXBASE)/../../products.mak))

include $(EXBASE)/../../products.mak/

# Define IPC\_INSTALL\_DIR since not defined in IPC top-level products.mak

IPC\_INSTALL\_DIR = $(word 1,$(subst /examples, examples,$(CURDIR)))

endif

endif

# By default, the necessary build variables are found/assigned via

# ../products.mak or ../../products.mak, included above. If you want to

# override these variables, or are building this example without

# ../products.mak or ../../products.mak, uncomment and assign the variables

# below.

DEPOT = /opt/ti

#### Linux toolchain ####

TOOLCHAIN\_LONGNAME = arm-linux-gnueabihf

TOOLCHAIN\_INSTALL\_DIR = /home/shankari/workdir/keystone/processor\_sdk\_02\_00\_02\_11/gcc-linaro-4.9-2015.05-x86\_64\_arm-linux-gnueabihf

TOOLCHAIN\_PREFIX = $(TOOLCHAIN\_INSTALL\_DIR)/bin/$(TOOLCHAIN\_LONGNAME)-

#DEPOT = \_your\_depot\_folder\_

#### Linux toolchain ####

#TOOLCHAIN\_LONGNAME = arm-none-linux-gnueabi

#TOOLCHAIN\_INSTALL\_DIR = $(DEPOT)/\_your\_linux\_gcc\_toolchain\_install\_

#TOOLCHAIN\_PREFIX = $(TOOLCHAIN\_INSTALL\_DIR)/bin/$(TOOLCHAIN\_LONGNAME)-

#### BIOS-side dependencies ####

#BIOS\_INSTALL\_DIR = $(DEPOT)/\_your\_bios\_install\_

#IPC\_INSTALL\_DIR = $(DEPOT)/\_your\_ipc\_install\_

#XDC\_INSTALL\_DIR = $(DEPOT)/\_your\_xdctools\_install\_

#### BIOS-side toolchains ####

#gnu.targets.arm.A15F = $(DEPOT)/\_your\_gnu\_arm\_codegen\_install\_

#ti.targets.elf.C66 = $(DEPOT)/\_your\_ti\_dsp\_codegen\_install\_

BIOS\_INSTALL\_DIR = /opt/ti/bios\_6\_45\_01\_29

IPC\_INSTALL\_DIR = /opt/ti/ipc\_3\_42\_00\_02

XDC\_INSTALL\_DIR = /opt/ti/xdctools\_3\_32\_00\_06\_core

#### BIOS-side toolchains ####

gnu.targets.arm.A15F = /opt/ti/ccsv6/tools/compiler/gcc-arm-none-eabi-4\_8-2014q3

ti.targets.elf.C66 = /opt/ti/ccsv6/tools/compiler/ti-cgt-c6000\_8.1.0

# Use this goal to print your product variables.

.show:

@echo "HOST TOOLCHAIN\_PREFIX = $(TOOLCHAIN\_PREFIX)"

@echo "BIOS\_INSTALL\_DIR = $(BIOS\_INSTALL\_DIR)"

@echo "IPC\_INSTALL\_DIR = $(IPC\_INSTALL\_DIR)"

@echo "XDC\_INSTALL\_DIR = $(XDC\_INSTALL\_DIR)"

@echo "gnu.targets.arm.A15F = $(gnu.targets.arm.A15F)"

@echo "ti.targets.elf.C66 = $(ti.targets.elf.C66)"

1. Build the example

**make clean**

**make**

The two binaries server\_core0.xe66 and app\_host will be generated.

### Copy the ex02\_messageq into the target file system.

Copy server\_core0.xe66 and app\_host into the filesystem

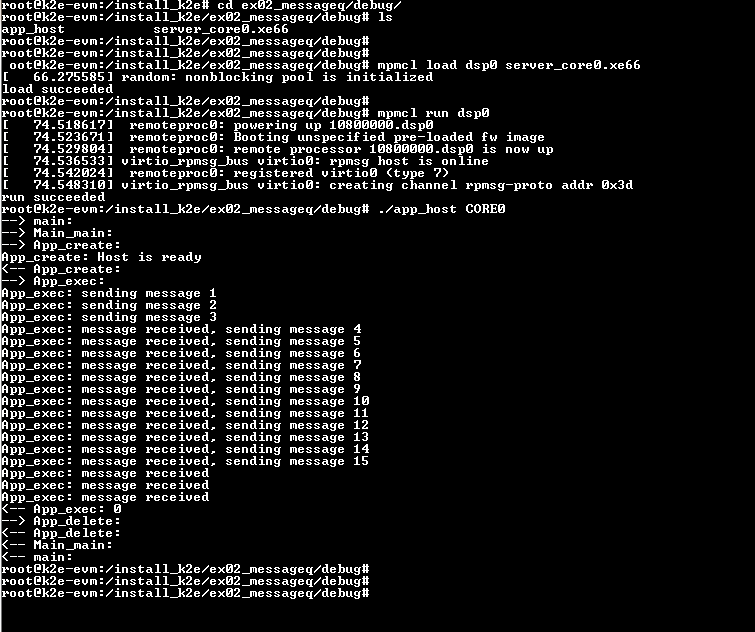
### Execute the ex02\_messageq example on the target EVM.

mpmcl load dsp0 server\*.xe66

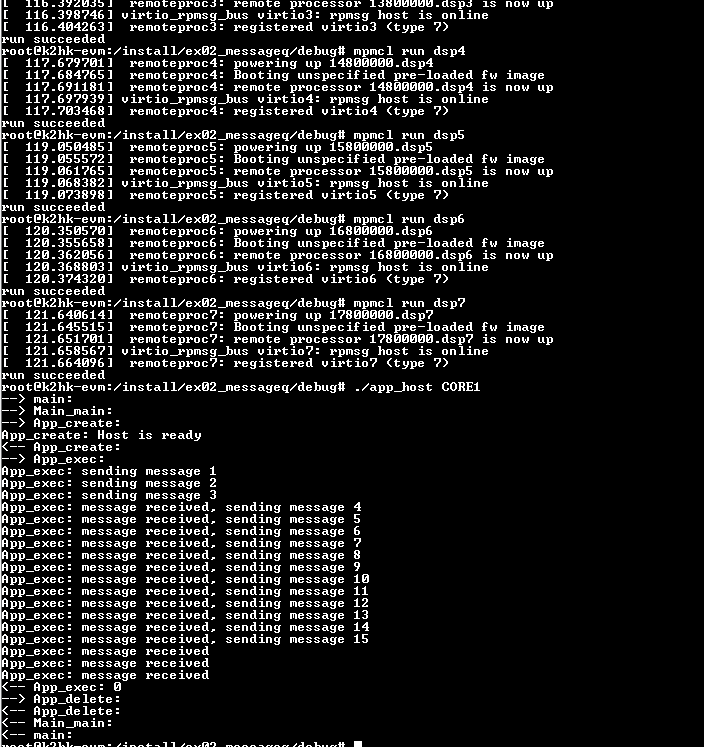
mpmcl run dsp0

./app\_host CORE0

### Output Logs- K2E EVM



### Output Logs- K2H EVM



# Lab 3: How to build and run ex45\_host on target

### Build the ex45\_host example in a host Ubuntu machine.

1. Create a work folder on your file system.

**mkdir work**

2. Extract this example into your work folder.

**cd work**

**unzip ex45\_host.zip**

3. Setup the build environment. Edit products.mak and set the install paths

as defined by your physical development area. Each example has its own

products.mak file; you may also create a products.mak file in the parent

directory which will be used by all examples.

edit ex45\_compute/products.mak

# look for other products.mak file to override local settings

ifneq (,$(wildcard $(EXBASE)/../products.mak))

include $(EXBASE)/../products.mak

else

ifneq (,$(wildcard $(EXBASE)/../../products.mak))

include $(EXBASE)/../../products.mak/

# Define IPC\_INSTALL\_DIR since not defined in IPC top-level products.mak

IPC\_INSTALL\_DIR = $(word 1,$(subst /examples, examples,$(CURDIR)))

endif

endif

# By default, the necessary build variables are found/assigned via

# ../products.mak or ../../products.mak, included above. If you want to

# override these variables, or are building this example without

# ../products.mak or ../../products.mak, uncomment and assign the variables

# below.

#DEPOT = \_your\_depot\_folder\_

DEPOT = /opt/ti

#### Linux toolchain ####

TOOLCHAIN\_LONGNAME = arm-linux-gnueabihf

TOOLCHAIN\_INSTALL\_DIR = /home/shankari/workdir/keystone/processor\_sdk\_02\_00\_02\_11/gcc-linaro-4.9-2015.05-x86\_64\_arm-linux-gnueabihf

TOOLCHAIN\_PREFIX = $(TOOLCHAIN\_INSTALL\_DIR)/bin/$(TOOLCHAIN\_LONGNAME)-

#### BIOS-side dependencies ####

BIOS\_INSTALL\_DIR = $(DEPOT)/bios\_6\_45\_01\_29

IPC\_INSTALL\_DIR = $(DEPOT)/ipc\_3\_42\_00\_02

XDC\_INSTALL\_DIR = $(DEPOT)/xdctools\_3\_32\_00\_06\_core

DESTDIR = $(DEPOT)/ipc\_3\_42\_00\_02/IPC\_install

#### BIOS-side toolchains ####

#ti.targets.elf.C66 = $(DEPOT)/\_your\_ti\_dsp\_codegen\_install\_

ti.targets.elf.C66 = $(DEPOT)/ccsv6/tools/compiler/ti-cgt-c6000\_8.1.0

# Use this goal to print your product variables.

.show:

@echo "TOOLCHAIN\_LONGNAME = $(TOOLCHAIN\_LONGNAME)"

@echo "TOOLCHAIN\_INSTALL\_DIR = $(TOOLCHAIN\_INSTALL\_DIR)"

@echo "TOOLCHAIN\_PREFIX = $(TOOLCHAIN\_PREFIX)"

@echo "BIOS\_INSTALL\_DIR = $(BIOS\_INSTALL\_DIR)"

@echo "IPC\_INSTALL\_DIR = $(IPC\_INSTALL\_DIR)"

@echo "XDC\_INSTALL\_DIR = $(XDC\_INSTALL\_DIR)"

@echo "DESTDIR = $(DESTDIR)"

@echo "ti.targets.elf.C66 = $(ti.targets.elf.C66)"

Note: To build this example, you must install IPC into DESTDIR.

4. Build the example. This will build only debug versions of the executables.

Edit the lower makefiles and uncomment the release goals to build both

debug and release executables.

**cd ex45\_host**

**make**

5. Issue the following commands to clean your example.

**cd ex45\_host**

**make clean**

### Copy the ex45\_host into the target file system.

Copy the HOST executables, and the supporting scripts to your target file system.

ex45\_host/host/bin/debug/thing1

ex45\_host/host/bin/debug/thing2

ex45\_host/scripts/run\_all.sh

ex45\_host/scripts/run\_lad.sh

### Execute the ex45\_host example on the target EVM.

A typical way to run this example is to run thing2 in the background and

then to run thing1 in the foreground. But this is not required. You can

run thing1 first if you prefer. It will spin in a loop trying to open the

message queue created by thing2. Once the queue is created by thing2, both

programs proceed as usual.

1. Start LAD. You must start the LAD daemon before running any IPC

program. If it is not already running, use the following command

to start it.

**lad\_tci6638 -l log.txt**

You can also use the helper script provided with the example.

**run\_lad.sh**

Note: The LAD daemon is built with the IPC product and is available

in the DESTDIR folder. Copy it to your target file system.

DESTDIR/bin/lad\_tci6638

2. You must run both thing1 and thing2 concurrently. To do this from

one shell, run the first program in the background. It does not matter

which one is run first. Then run the second program in the foreground.

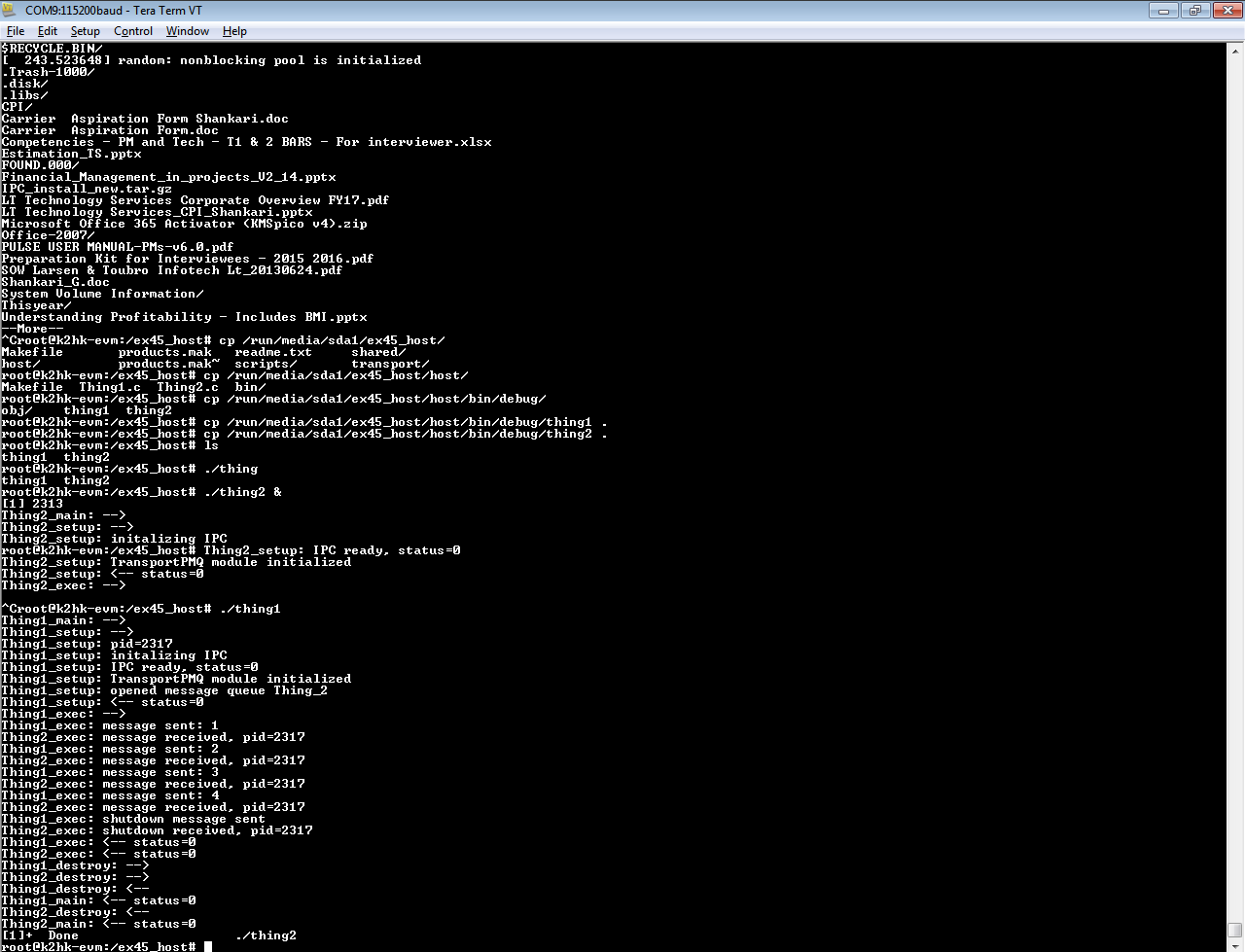
**thing2 &**

**thing1**

You can also use the helper script provided with the example.

**run\_all.sh**

### Output Logs



# Lab 4: How to build and run ex46\_graph on target

### Build the ex46\_graph example in a host Ubuntu machine.

1. Create a work folder on your file system.

**mkdir work**

2. This example uses the transport built in the ex45\_host example. You

must install and build the ex45\_host example first.

**cd work**

**unzip ex45\_host.zip**

Follow the instructions in ex45\_host/readme.txt to build the example.

3. Extract this example into your work folder.

**cd work**

**unzip ex46\_graph.zip**

4. Setup the build environment. Edit products.mak and set the install paths

as defined by your physical development area. Each example has its own

products.mak file; you may also create a products.mak file in the parent

directory which will be used by all examples.

edit ex46\_graph/products.mak

# look for other products.mak file to override local settings

ifneq (,$(wildcard $(EXBASE)/../products.mak))

include $(EXBASE)/../products.mak

else

ifneq (,$(wildcard $(EXBASE)/../../products.mak))

include $(EXBASE)/../../products.mak/

# Define IPC\_INSTALL\_DIR since not defined in IPC top-level products.mak

IPC\_INSTALL\_DIR = $(word 1,$(subst /examples, examples,$(CURDIR)))

endif

endif

# By default, the necessary build variables are found/assigned via

# ../products.mak or ../../products.mak, included above. If you want to

# override these variables, or are building this example without

# ../products.mak or ../../products.mak, uncomment and assign the variables

# below.

#DEPOT = \_your\_depot\_folder\_

DEPOT = /opt/ti

#### Linux toolchain ####

TOOLCHAIN\_LONGNAME = arm-linux-gnueabihf

TOOLCHAIN\_INSTALL\_DIR = /home/shankari/workdir/keystone/processor\_sdk\_02\_00\_02\_11/gcc-linaro-4.9-2015.05-x86\_64\_arm-linux-gnueabihf

TOOLCHAIN\_PREFIX = $(TOOLCHAIN\_INSTALL\_DIR)/bin/$(TOOLCHAIN\_LONGNAME)-

#### BIOS-side dependencies ####

BIOS\_INSTALL\_DIR = $(DEPOT)/bios\_6\_45\_01\_29

IPC\_INSTALL\_DIR = $(DEPOT)/ipc\_3\_42\_00\_02

XDC\_INSTALL\_DIR = $(DEPOT)/xdctools\_3\_32\_00\_06\_core

DESTDIR = $(DEPOT)/ipc\_3\_42\_00\_02/IPC\_install

#### BIOS-side toolchains ####

#ti.targets.elf.C66 = $(DEPOT)/\_your\_ti\_dsp\_codegen\_install\_

ti.targets.elf.C66 = $(DEPOT)/ccsv6/tools/compiler/ti-cgt-c6000\_8.1.0

# Use this goal to print your product variables.

.show:

@echo "TOOLCHAIN\_LONGNAME = $(TOOLCHAIN\_LONGNAME)"

@echo "TOOLCHAIN\_INSTALL\_DIR = $(TOOLCHAIN\_INSTALL\_DIR)"

@echo "TOOLCHAIN\_PREFIX = $(TOOLCHAIN\_PREFIX)"

@echo "BIOS\_INSTALL\_DIR = $(BIOS\_INSTALL\_DIR)"

@echo "IPC\_INSTALL\_DIR = $(IPC\_INSTALL\_DIR)"

@echo "XDC\_INSTALL\_DIR = $(XDC\_INSTALL\_DIR)"

@echo "DESTDIR = $(DESTDIR)"

@echo "ti.targets.elf.C66 = $(ti.targets.elf.C66)"

Note: To build this example, you must install IPC into DESTDIR.

5. Build the example. This will build only debug versions of the executables.

Edit the lower makefiles and uncomment the release goals to build both

debug and release executables.

**cd ex46\_graph**

**make**

6. Issue the following commands to clean your example.

**cd ex46\_graph**

**make clean**

### Copy the ex46\_graph into the target file system.

Copy the executables and supporting scripts to your target file system.

ex46\_graph/combiner/bin/debug/combinerN.xe66

ex46\_graph/consumer/bin/debug/consumer

ex46\_graph/manager/bin/debug/manager

ex46\_graph/producer/bin/debug/producer

ex46\_graph/scripts/patchExec.pl

ex46\_graph/scripts/run\_lad.sh

ex46\_graph/scripts/run\_patch\_combiner.sh

ex46\_graph/scripts/run\_patch\_transformer.sh

ex46\_graph/scripts/vritio.awk

ex46\_graph/transformer/bin/debug/transformerN.xe66

### Execute the ex46\_graph example on the target EVM

k2hk-evm login: root

root@k2hk-evm:~# cd ex46\_graph/

root@k2hk-evm:~/ex46\_graph# ps -e | grep lad

1938 ? 00:00:00 lad\_tci6638

root@k2hk-evm:~/ex46\_graph# kill 1938

root@k2hk-evm:~/ex46\_graph# **./lad\_tci6638 -r 8 -n 9 -b 0 -l log.txt**

Set LAD's synchronization scheme to ProcSync\_PAIR

Set LAD's number of processors to 4608

Set LAD's base cluster id to 0

Set LAD's number of reserved queues to 8

Opened log file: log.txt

numProcessors = 4608 id = 0 baseId = 0

Spawned daemon: lad\_tci6638

root@k2hk-evm:~/ex46\_graph# ls

combinerN.xe66 patchExec.pl run\_patch\_combiner.sh

combinerN\_p.xe66 producer run\_patch\_transformer.sh

consumer run\_dsp\_graph.sh transformerN.xe66

manager run\_lad.sh transformerN\_p.xe66

root@k2hk-evm:~/ex46\_graph# ./manager

[cm]> status

Program Status:

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

- -------- dsp1: cqid=0x0000ffff

- -------- dsp2: cqid=0x0000ffff

- -------- dsp3: cqid=0x0000ffff

- -------- dsp4: cqid=0x0000ffff

- -------- dsp5: cqid=0x0000ffff

- -------- dsp6: cqid=0x0000ffff

- -------- dsp7: cqid=0x0000ffff

- -------- dsp8: cqid=0x0000ffff

Connections:

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

[cm]> launch producer

[cm]> launch consumer

[cm]> launch transformer dsp1

load succeeded

[ 154.177716] remoteproc0: powering up 10800000.dsp0

[ 154.182615] remoteproc0: Booting unspecified pre-loaded fw image

[ 154.189274] remoteproc0: remote processor 10800000.dsp0 is now up

[ 154.196131] virtio\_rpmsg\_bus virtio0: rpmsg host is online

[ 154.196205] virtio\_rpmsg\_bus virtio0: creating channel rpmsg-proto addr 0x3d

[ 154.209564] remoteproc0: registered virtio0 (type 7)

run succeeded

[cm]> status

Program Status:

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

A producer host: cqid=0x0000008a off

B consumer host: cqid=0x0000008b

C transformer dsp1: cqid=0x00010088

- -------- dsp2: cqid=0x0000ffff

- -------- dsp3: cqid=0x0000ffff

- -------- dsp4: cqid=0x0000ffff

- -------- dsp5: cqid=0x0000ffff

- -------- dsp6: cqid=0x0000ffff

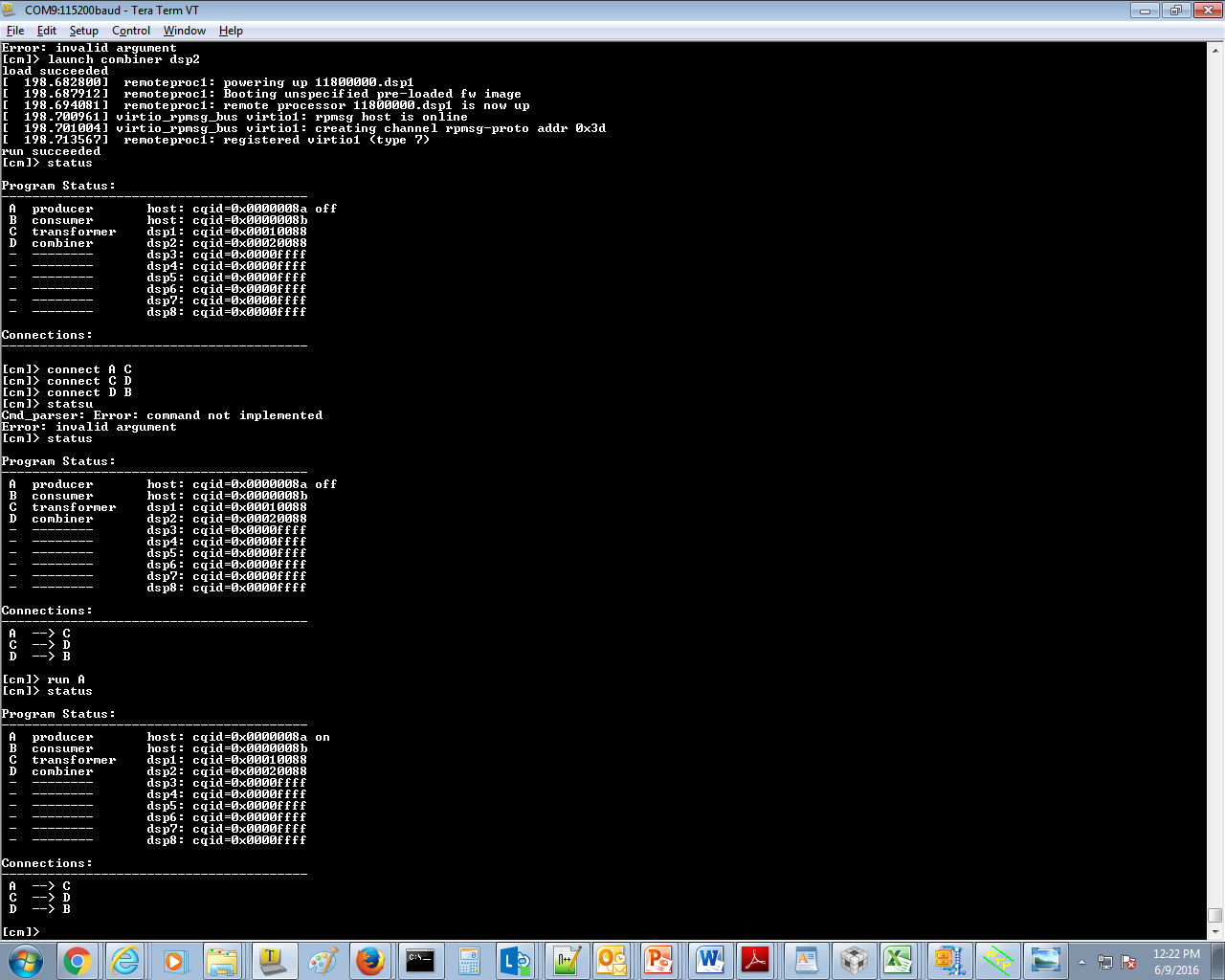
- -------- dsp7: cqid=0x0000ffff

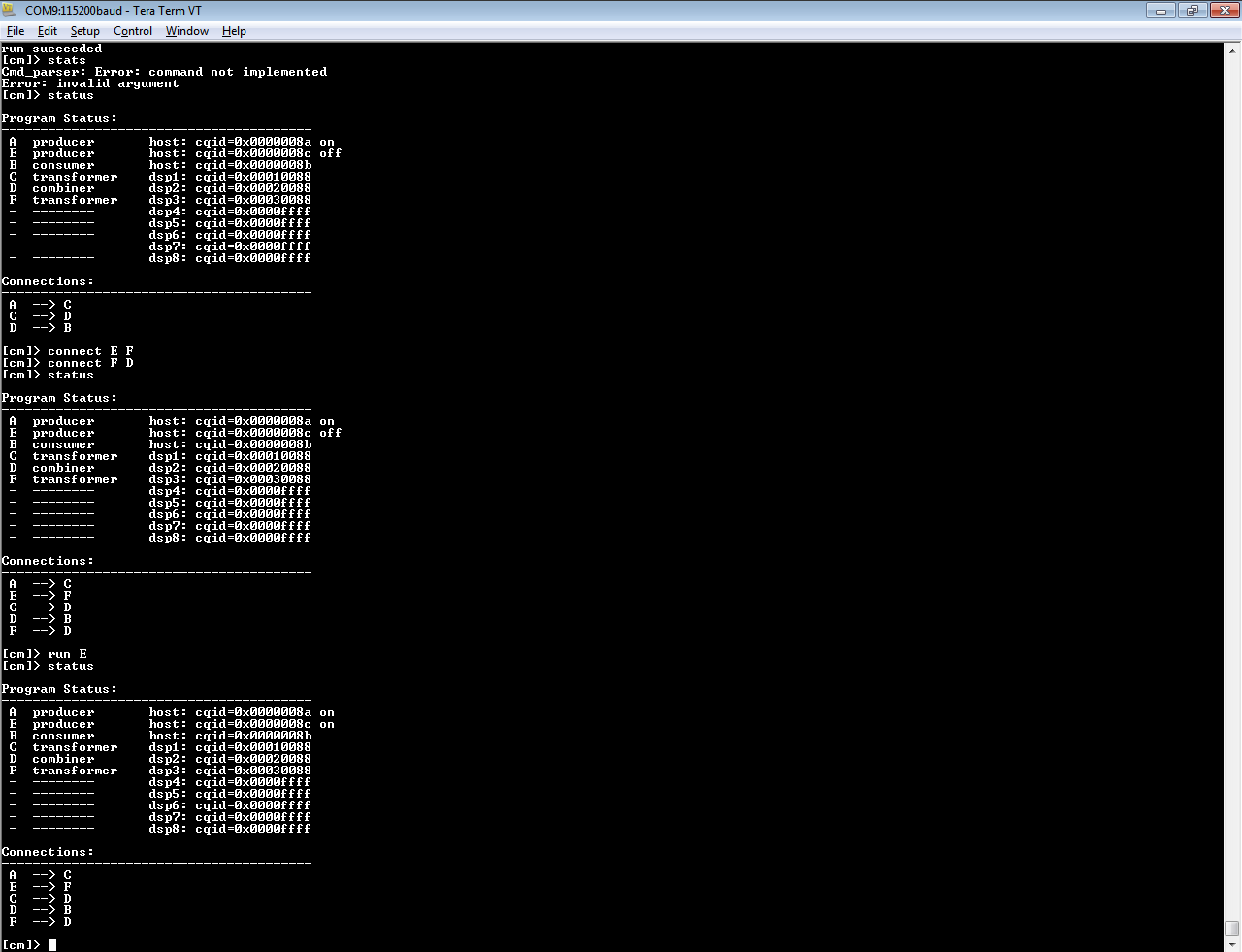
- -------- dsp8: cqid=0x0000ffff

Connections:

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

### Output Logs





# Debugging steps for DSP side programs for the IPC examples.

1. mpmcl load dsp0 server\*.xe66
2. mpmcl run dsp0
3. Do not make any changes in the BOOT switch settings

Launch an empty K2E cxml file without including any gel script

connect target

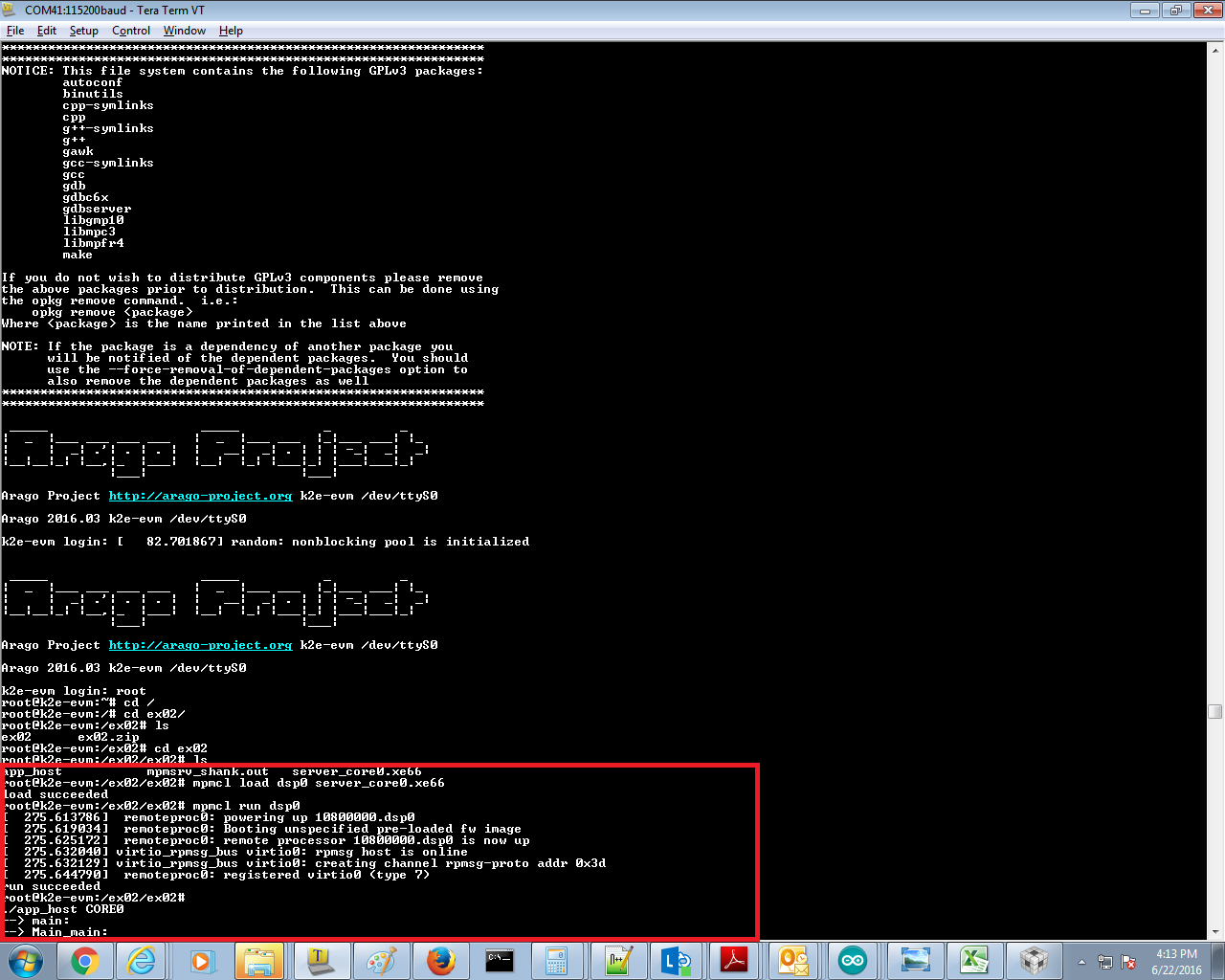
Load --> load symbols --> select server\*.xe66

Browsing the map file and put breakpoints where you would like to break.

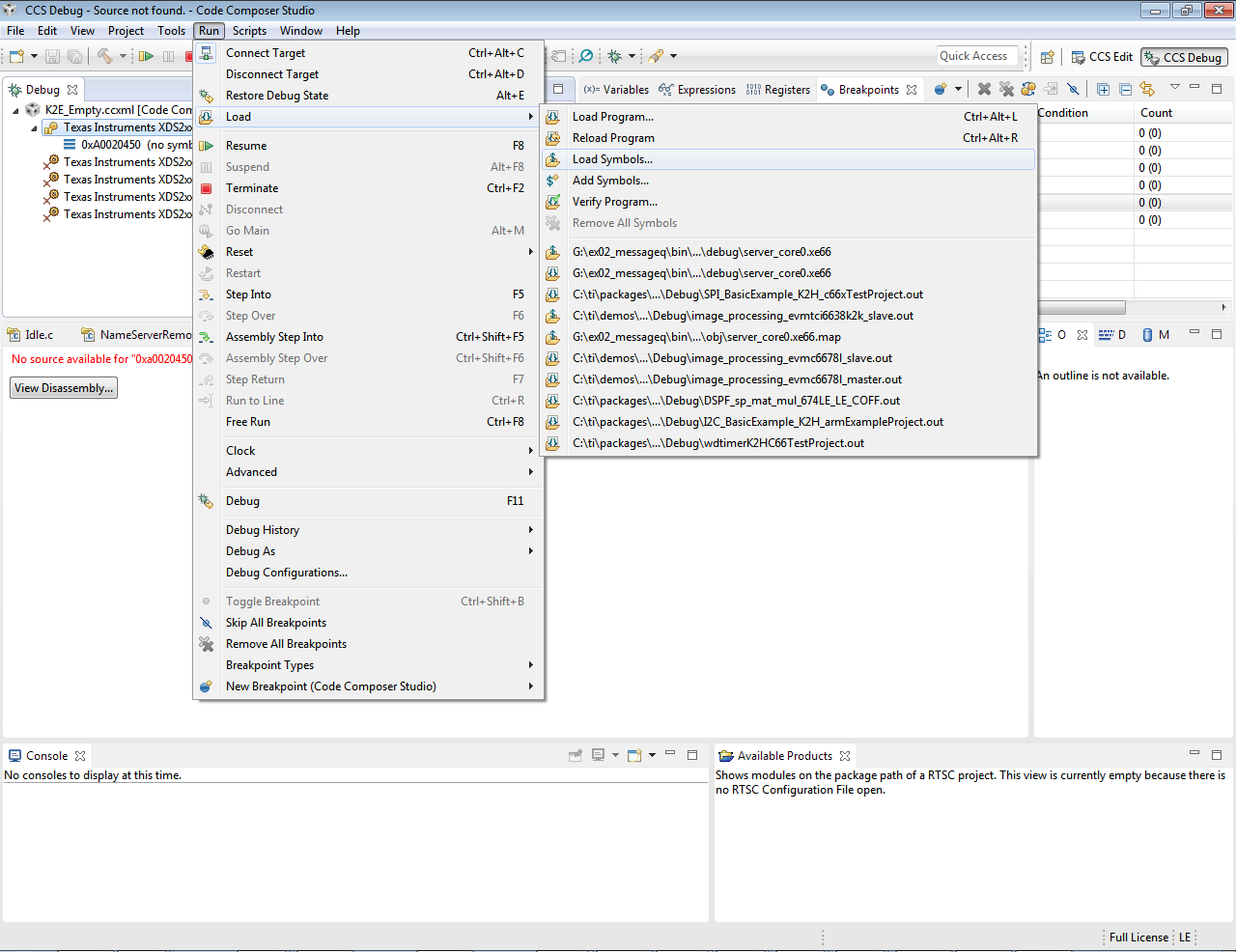
run

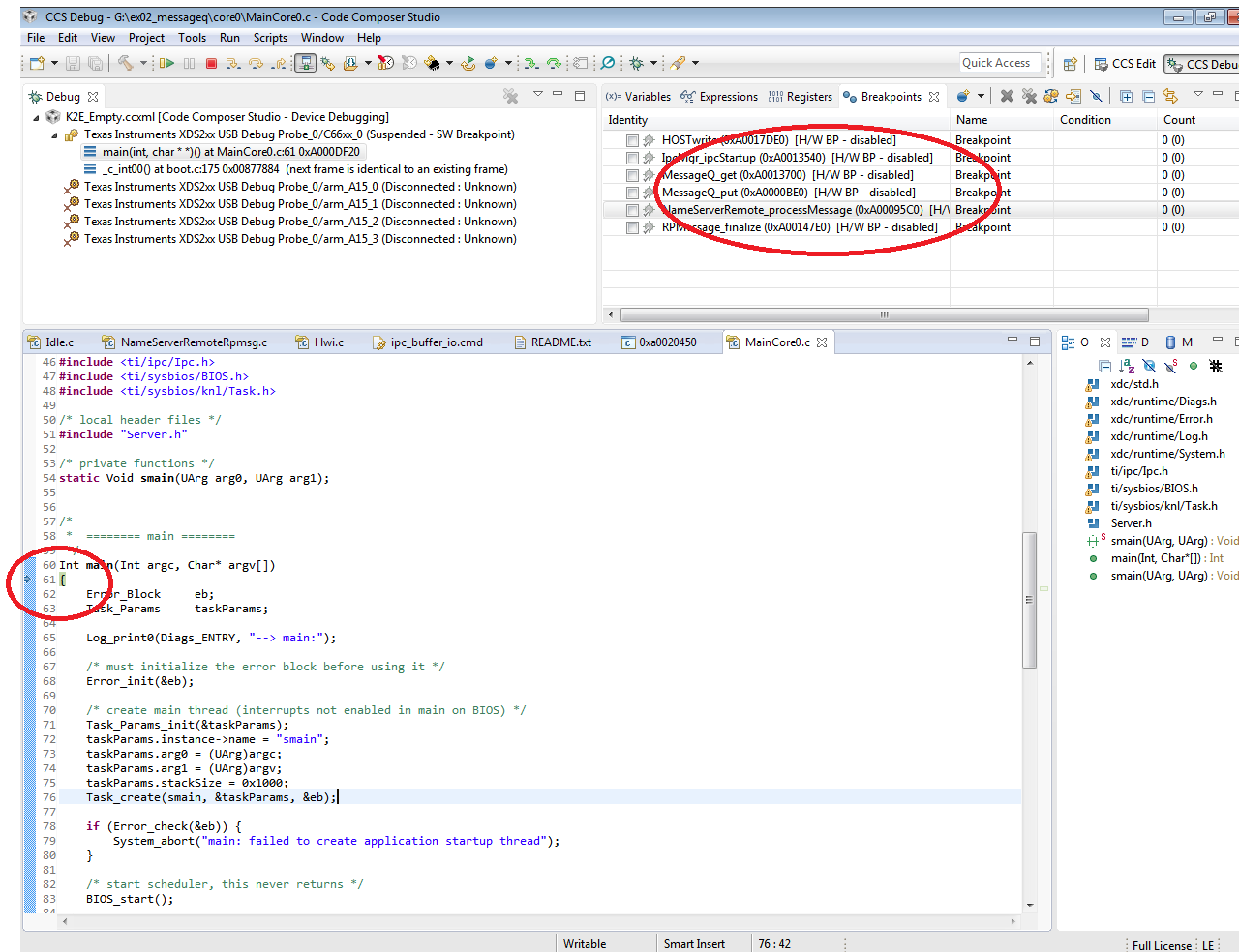
In the teraterm ( Linux machine ) launch the app\_host

./app\_host CORE0

****

Load the DSP executable using the load symbol option

****

****

**======================END========================**