Keystone II guide on running IPC examples
Contents

Prerequisites .................................................................................................................................................. 1

Hardware ................................................................................................................................................... 1

Software .................................................................................................................................................... 1

1. Updating the U-BOOT .......................................................................................................................... 2
   Update SPI NOR Flash with U-boot GPH image .................................................................................. 2

2. Set the BOOT switch settings as SPI boot mode and boot the U-boot ............................................ 3

3. Flash UBIFS into NAND and boot Linux .......................................................................................... 4

4. Build IPC package .............................................................................................................................. 9

Lab 1: How to build and run ex44_compute on target .............................................................................. 11

   Build the ex44_compute example in a host Ubuntu machine ............................................................. 11
   Copy the ex44_compute into the target filesystem .............................................................................. 12
   Execute the ex44_compute example on the target EVM ................................................................. 13
   Output Logs ................................................................................................................................... 13

Lab 2: How to build and run ex02_MessageQ on target ............................................................................. 18

   Build the ex02_messageq example in a host Ubuntu machine .......................................................... 18
   Copy the ex02_messageq into the target filesystem .......................................................................... 19
   Execute the ex02_messageq example on the target EVM ................................................................. 19
   Output Logs- K2E EVM ................................................................................................................ 20
   Output Logs- K2H EVM ............................................................................................................... 20

Lab 3: How to build and run ex45_host on target ..................................................................................... 21

   Build the ex45_host example in a host Ubuntu machine ................................................................. 21
   Copy the ex45_host into the target file system ................................................................................. 23
   Execute the ex45_host example on the target EVM ......................................................................... 23
   Output Logs ................................................................................................................................... 24

Lab 4: How to build and run ex46_graph on target .................................................................................. 25

   Build the ex46_graph example in a host Ubuntu machine ............................................................... 25
   Copy the ex46_graph into the target file system ............................................................................... 27
   Execute the ex46_graph example on the target EVM ....................................................................... 27
Output Logs .............................................................................................................................................. 30
Debugging steps for DSP side programs for the IPC examples............................................................... 33
**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.
   

2. EVM k2E Revision 1.0.2.2 and above.
   

**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
   
   

2. 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.
1. 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.

2. 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

3. 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`
Prerequisites

(Offset is depends on u-boot image size)

sf erase 0 0x100000

sf write 0x88000000 0 0x100000

2. 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

<table>
<thead>
<tr>
<th>Pin#</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>OFF</td>
<td>OFF</td>
<td>ON</td>
<td>OFF</td>
</tr>
</tbody>
</table>

Do a power on reset and observe that the newly update u-boot is up and running
3. **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
```
Prerequisites

KeyStone Multicore Workshop
Prerequisites

---

2.1 Kilos

Bytes transferred = 410386432 (187600000 hex)

$Boot# nand erase.part ubifs

NAND erase.part: device 0 offset 0x100000, size 0x1fe00000
Skipping bad block at 0x1fff8000
Skipping bad block at 0x1ffe0000
Skipping bad block at 0x1fff0000

OK

$Boot# pre burn_uhi
burn_uhi=nand erase.part ubifs; nand write $(addr_uhi) ubifs $(filesize)

$Boot# run burn_uhi

NAND erase.part: device 0 offset 0x100000, size 0x1fe00000
Skipping bad block at 0x1fff8000
Skipping bad block at 0x1ffe0000
Skipping bad block at 0x1fff0000

OK

NAND write: device 0 offset 0x100000, size 0x18760000
410386432 bytes written: OK

$Boot# env default -a

# Resetting to default environment
$Boot# setenv boot ubi
$Boot# saveenv

Saving environment to NAND...

Erasing NAND...
Erasing at 0x120000 -- 100% complete.
Writing to NAND... OK

$Boot#
Prerequisites

KeyStone Multicore Workshop
5. When booting ends, login as root (no password)
4. **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
   ```

2. Create a new directory to store the IPC libraries:
   
   ```
   mkdir IPC_install
   ```

3. 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+bbe8ccfca-gbbe8ccfc
   # 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 =

4. 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.

```bash
mkdir work
```

2. Extract the example located inside the IPC package.

```bash
cd work
unzip ex44_compute.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.

```makefile
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/
    endif
endif

# Define IPC_INSTALL_DIR since not defined in IPC top-level products.mak
IPC_INSTALL_DIR = $(word 1,$(subst /examples, examples,$(CURDIR)))
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)"
```
Lab 8: ARM Optimization Using SMP Linux

@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)"

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 ex44_compute
   make

5. 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 filesystem.

   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

2. 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

3. Rebuild your LAD executable. Copy the LAD executable to your target file system.
4. Copy to IPC_INSTALL_DIR folder your filesystem.

**Execute the ex44_compute example on the target EVM.**

1. Go to IPC_install/bin and launch the Lad daemon.

   ```bash
   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/

   ```bash
   run run_dsp.sh
   ```

3. Goto host folder

   ```bash
   ./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: initializing IPC
App_setup: initializing 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 PEB queue: qid=0x00010082

App_exec: message sent to PEB queue: qid=0x00020082

App_exec: message sent to PEB queue: qid=0x00030082

App_exec: message sent to PEB 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: 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

=====================================================================
# below.

DEPOT = /opt/ti

#### Linux toolchain ####

#TOOLCHAIN_LONGNAME     = arm-linux-gnueabihf
#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_

# 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)"

3. 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
Lab 8: ARM Optimization Using SMP Linux

/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.
   
   ```sh
   mkdir work
   ```

2. This example uses the transport built in the ex45_host example. You must install and build the ex45_host example first.

   ```sh
   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/
  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)/xctools_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)"
```
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.

   ```bash
   cd ex46_graph
   make
   ```

6. Issue the following commands to clean your example.

   ```bash
   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 ProcSyncPAIR

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