DSP/BIOS™ Link

Migration Guide

1.61

Published on 9th December 2008

IMPORTANT NOTICE

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address:

Texas Instruments,
Post Office Box 655303,
Dallas, Texas 75265

Table of Contents

List of Tables

Read This First

About This Manual

This document describes how to migrate to the new DSP/BIOS™ Link.

How to Use This Manual

This document includes the following chapters:

- Chapter 1, *Migration* - describes the steps required to migrate to new DSP/BIOS™ Link.

Please go through the Release Notes document available in the release package before starting the installation.

Notation of information elements

The document may contain these additional elements:

!

Warning

This is an example of warning message. It usually indicates a non-recoverable change.

!

Caution

This is an example of caution message.

!

Important

This is an example of important message.

!

Note

This is an example of additional note. This usually indicates additional information in the current context.

Tip

This is an example of a useful tip.

If You Need Assistance

For any assistance, please send an mail to software support.

Trademarks

DSP/BIOS™ is a trademark of Texas Instruments Incorporated.

All other trademarks are the property of the respective owner.

Migration

Abstract

This chapter describes how to migrate to the new DSP/BIOS™ Link.

Table of Contents

1.1. Introduction

1.1.1. Purpose and Scope

This document describes how to migrate to the new DSP/BIOS™ Link version 1.61GA from 1.51 GA .

DSP/BIOS LINK provides communication and control infrastructure between GPP and DSP and is aimed at traditional embedded applications. Many applications require a specific framework for communication and control between GPP and DSP.

The document does not discuss the packaging and installation.

The development teams for DSP/BIOS LINK, System integarators and Application developers are the intended audience of this document.

1.1.2. Terms and Abbreviations

GPP	General Purpose Processor
DSP	Digital Signal processor
OS	operating system
DSPLink	A generic term used for DSP/BIOS™ Link.

Table 1.1. Terms and Abbreviations

1.1.3. References

LNK 137 DES	DYNAMIC CONFIGURATION
LNK 182 DES	Multi-DSP Design Document

Table 1.2. References

1.2. Overview

DSP/BIOS™ Link is runtime software, analysis tools, and an associated porting kit that simplifies the development of embedded applications in which a general-purpose microprocessor controls and communicates with a TI DSP. DSP/BIOS™ Link provides control and communication paths between GPP OS threads and DSP/BIOS™ tasks, along with analysis instrumentation and tools.

This document outlines the steps required to migrate from previous version of DSPLink to the multi-DSP DSPLink.

1.3. Static Configuration

This section describes changes that were made to static configuration step.

Script has been upgraded to take configuration options from command line instead of menu driven interface. This is done so that it can be made scriptable. Also the internal logic of this script is updated so, that porters can easily add new platforms to the this script.

1.3.1. Usage

The updated script now takes configuration options from the command line. Below gives the options it takes:

Platform	Base Platform to used by DSPLink. For example: --platform=DAVINCI
No of DSPs	Number of dsps to controlled by the DSPLink. For example: --nodsp=1
DSPCFG#	DSP to be controlled (DSP procId #) & Physical Interface. For example: --dspcfg_0=DM6446GEMSHMEM
DSP OS#	DSP OS used by DSP#. For example: --dspos_0=DSPBIOS5XX
GPP OS	GPP OS to be used by DSPLink. For example: --gppos=MVL4G
COMPONENTS	Components/Modules to be used by DSPLink. For example: --comps=lmrc
TRACE	Enable tracing for DSPLink (optional). For example: --trace=1
GPP Temp Directory	Temporary path for GPP binaries and libraries generation (optional). For example: --gpp_temp=/tmp/gpptemp
DSP# Temp Dir	Temporary path for DSP binaries and libraries generation (optional). For example: --dsp1_temp=/tmp/dsp1temp
Leagcy	Enable legacy support (optional). For example: --legacy=1
Filesystem	Filesystem type to be used by GPP OS (exists only on few platform). For example: --fs=PSEUDOFS
dspdma	This option is to override the default memcpy option with DMA(using DSP EDMA) to do data transfer between host and DSP. This option is valid only for LINUXPC platforms.This is optional argument and if not specified default memcpy is used for data transfer. For example: --dspdma=1

Table 1.3. Usage

For example, if DAVINCI platform is need to be configured with 1 DSP, DSP/BIOS™ version 5.XX, GPP OS as montavista pro 4.0 with glibc libraries (and RingIO, CHNL, MSGQ, MPLIST modules) then type the following:

```
perl $DSPLINK_RPOOT/config/bin/dsplinkcfg.pl --platform=DAVINCI --nodsp=1 --
dspcfg_0=DM6446GEMSHMEM --dspos_0=DSPBIOS5XX --gppos=MVL4G --comps=lmrc
```

1.3.2. Output of the script

Note

All exported enviorment variables are prefixed with TI_DSPLINK_, so that user's workspace is not clutterred.

Also this script generates all compiler and linker related defines (for example: TI_DSPLINK_MAX_DSPS) inside the CURRENTCFG.MK file. This defines are generated without -D/-d so that user can easily port it their compiler/linker. This can be done very easily in gmake based system:

```
$(addprefix -D, $(TI_DSPLINK_GPP_DEFINES))
```

Result of above script is some files, as explained below:

1. CFG_system.c
2. multimake.bat
3. RTSC files

.1. CFG_system.c

This file contains the overall configured architecture of the DSPLink. Typical contents of this file is as follows:

```
LINKCFG_Object LINKCFG_Config = {
 &LINKCFG_gppObject,
 {&DM6446GEM_SHMEM_Config, },
} ;
```

LINKCFG_gppObject is pointer to the GPP configuration object, contains parameters for configuring GPP. DM6446GEM_SHMEM_Config is pointer to DSP configuration object, contains parameters for configuring DSP. There can be one or more DSP objects as configured through the static configuration step. Users can pass LINKCFG_Config to override the default configuration values through PROC_setup. For example:

```
PROC_setup (&LINKCFG_Config) ;
```

.2. multimake.bat

This file contains a script to build DSP side sources for all DSPs configured. If user issues a make command instead of this script, only first DSP will be built. Location of this file is

```
${DSPLINK}/etc/host/scripts/Linux/multimake.sh
```

for Linux and for Windows it is

```
${DSPLINK}\etc\host\scripts\msdos\multimake.bat
```

Invokation example:

```
$multimake.sh debug -s
```

.3. RTSC files

RTSC packaging is updated to reflect paths and configuration for default DSP (i.e DSP with procId = 0). GPP side packaging remains the same.

1.4. Dynamic configuration

This section describes changes made to dynamic configuration feature.

1.4.1. Overview

Dynamic configuration of DSPLink is now split into two steps:

1.4.1.1. GPP related

GPP related config values are read and processed at the time of PROC_setup. To support multi-applications, DSP values are read at the time of PROC_setup but not processed. they are processed at the time of PROC_attach. Typical values of GPP configuration is:

```
   LINKCFG_Gpp  LINKCFG_gppObject = {
   "ARM9",                                   /* NAME            : Name of the
GPP */
   16,                                       /* MAXMSGQS        : Maximum MSGQs
that can be opened */
   16,                                       /* MAXCHNLQUEUE    : Maximum Queue
Length for all channels */
   (Uint32) -1,                              /* POOLTABLEID     : ID of the POOL
table (-1 if not needed) */
   0,                                        /* NUMPOOLS        : Number of
POOLs supported */
   (LINKCFG_GppOs *) &LINKCFG_gppOsObject    /* GPPOSOBJECT     : Pointer to GPP
OS object */
   } ;
```

Most of these values are not changed frequently by the application writers.

1.4.1.2. DSP related

DSP related values are processed at the time of PROC_attach, this helps users to change some config parameters and reinitialize the DSP without calling PROC_destroy or without destroying other DSPs context.

Note

Applications with come and go type of DSP utilization should note that passing config values for the DSP is mandatory in PROC_attach, otherwise PROC_attach would report error.Even though If default configuraion is used attrs should be initialized to NULL before calling PROC_attch

---

Example of passing Config values in PROC_attach is as follows:

```
PROC_Attrs procAttrs ;
procAttrs.dspCfgPtr = DM6446GEM_SHMEM_Config ;
```

```
PROC_attach (DSPID, &procAttrs) ;
```

A simple way of changing config values for DSP is as follows (Suppose platform is DAVINCI):

1. Copy the CFG_DM6446GEM_SHMEM.c in your application src tree.
2. Modify the required parameters/values and rename the DM6446GEM_SHMEM_Config to DM6446GEM_SHMEM_AppConfig.
3. Compile this file with the application src tree.
4. Pass the DM6446GEM_SHMEM_AppConfig to PROC_attach as above said.

Four new arguments (ARG1,..) were added to LINKCFG_dspObject for each DSP supported DSPLink. These arguments are platform specific. On PCI Platforms:

1. ARG1 - Bus Number of PCI Cards
2. ARG2 - Slot Number of PCI Cards
3. ARG3 - DSPLink Shared memory region entry Number
4. ARG4 - Physical interface type (PCI_INTERFACE for PCI)

On VLYNQ Platforms:

1. ARG1 - Don't care
2. ARG2 - Don't care
3. ARG3 - DSPLink Shared memory region entry Number
4. ARG4 - Physical interface type (VLYNQ_INTERFACE for VLYNQ)

For platforms with shared memory interface (For exampple: DaVinci) these four argument are don't care.

1.4.1.3. POOL related

A new field POOLENTRYID has been added in POOL entry inside the static configuartion file to indicate the memory entry ID for the pool buffers. Control structures of pool are allocated from the DSPLink shared memory region, while the memory for buffers are allocated from the memory region pointed by POOLENTRYID. This memory region can provide buffer memories for one or more POOLs, depending upon the size of this region.

```
     STATIC LINKCFG_Pool  LINKCFG_poolTable_00 [] =
     {
      {
       "SMAPOOL",                /* NAME           : Name of the pool */
       (Uint32) SHAREDENTRYID1,  /* MEMENTRY       : Memory entry ID (-1 if not
needed) */
```

```
      (Uint32) 0x70000,         /* POOLSIZE       : Size of the pool (-1 if not
needed) */
      (Uint32) -1,              /* IPSID          : ID of the IPS used */
      (Uint32) -1,              /* IPSEVENTNO     : IPS Event number associated
with POOL */
      POOLENTRYID,              /* POOLMEMENTRY   : Pool memory region section
ID  */
      0x0,                      /* ARGUMENT1      : First Pool-specific
argument */
      0x0                       /* ARGUMENT2      : Second Pool-specific
argument */
     }
    } ;
```

The advantages of this are:

- The POOL entry size is now exactly equal to (buffer size * number of buffers), and can be thus optimized.
- All control structures in DSPLink are now within the DSPLINKMEM (and DSPLINKMEM1 if applicable), and the size of this region can be reduced to minimum requirement, which is usually fixed for a specific build configuration. This would not change with changes in application buffer requirements.
- For PCI and VLYNQ devices, this optimizes execution behavior, since DMA transfers are not required for short control operations.

For platforms, where GPP and DSP does not have any shared memory or where DSP's memory region are mapped through some transport interface such as PCI or VLYNQ, DSPLink allocates a big physical area for POOL buffer memory and keep it synchornized (Using DMA) with DSP memory. Here A new field SYNCD is added to memory region entries to indicate if the memory region is kept in synchornized with DSP memory. So memory region pointed by POOLENTRYID must have SYNCD field set to true, while all other memory regions must have set it to false.

```
    {
     SHAREDENTRYID0,                  /* ENTRY          : Entry number */
     "DSPLINKMEM",                    /* NAME           : Name of the memory
region */
     SHAREDMEMORYADDR0,               /* ADDRPHYS       : Physical address */
     SHAREDMEMORYADDR0,               /* ADDRDSPVIRT    : DSP virtual address
*/
     (Uint32) -1,                     /* ADDRGPPVIRT    : GPP virtual address
(if known) */
     SHAREDMEMORYSIZE0,               /* SIZE           : Size of the memory
region */
     TRUE,                            /* SHARED         : Shared access memory?
*/
     FALSE,                           /* SYNCD          : Synchornized? */
    },
    {
     POOLENTRYID,                      /* ENTRY          : Entry number */
     "POOLMEM",                        /* NAME           : Name of the memory
region */
```

```
    POOLMEMORYADDR,                     /* ADDRPHYS       : Physical address */
    POOLMEMORYADDR,                     /* ADDRDSPVIRT    : DSP virtual address
*/
    (Uint32) -1,                        /* ADDRGPPVIRT    : GPP virtual address
(if known) */
    POOLMEMORYSIZE,                     /* SIZE           : Size of the memory
region */
    TRUE,                               /* SHARED         : Shared access
memory? Logically */
    TRUE,                               /* SYNCD          : Synchornized? */
   },
```

1.5. GPP side sources

1.5.1. API

1.5.1.1. PROC

PROC module is now updated for multi-DSP application, the following changes have been made to it:

- To change dsp configuration dynamically, Applications need to pass the dsp configuration information through the PROC_attach API.
- If dynamic configuration is to be used in multi-processing/multi-application scenario, PROC_setup () (and correspondingly PROC_destroy ()) must now be called in all processes to pass the new dynamic configuration information. If this is not done, the other processes shall only get default configuration information, and updated dynamic configuration shall not be available in their user space. This may cause non-deterministic results.

1.5.1.2. POOL

- Now there two different type of POOL implementation available, SMAPOOL is used on the platform where shared memory is available and DMAPOOL for the platform with PCI or VLYNQ interface.
- PoolId interpretation has changed. PoolId (16 Bits) is now seen as upper 8 bits as processor Identifier and lower 8 bits as pool number in the given processor. PoolId is restricted to 16 bits, since DSP/BIOS™ has MSGQ/POOL protocol based on 16 bit poolId. For application user, calling

  ```
  POOL_makePoolId (procId, poolNo)
  ```

  gives the correct PoolId. So all the APIs which take poolId as argument or part of argumnet structure, now take the return value of the POOL_makePoolId as argumnet instead of poolId. When single-DSP configuration is used, then application does not need to change to call this extra macro. But if multi-DSP configuration is used, then this macro must be used.

Note

Maximun number of pools that can be configured is now restricted to 256 (since poolId is 8-bit)..

1.5.1.2.1. RingIO

RingIO module is now updated for multi-DSP application, the following changes have been made to it:

- RingIO_create function now takes Processor Identifier. So that a RingIO can be created between GPP and a DSP.

- RingIO_delete function now takes Processor Identifier.

But there is legacy support available as well, for legacy support users are required to pass the --legacy=1 option to the static configuration script.In legacy mode, applications will be able to use the the RingIO_create and RingIO_delete APIs with out the procId.

1.5.1.2.2. MSGQ

MSGQ module is now updated for multi-DSP application, the following changes have been made to it:

- In Multi dsp configurations, applications must update poolId of mqtAttrs structure with the value obtained from POOL_makePoolId and then need to call MSGQ_transportOpen API to open a transport to a specific DSP.
- In Multi dsp configurations, applications must pass the id got from POOL_makePoolId to MSGQ_alloc call to allocate a buffer from the pool that is configured for the perticular DSP.For example if application wants to send a message to DSP processor follow the below steps.

1.5.1.3. Misc

Previous DSPLink release used to have dynamic configuration files being copied to API directory. Now instead of API directory these files are copied ${DSPLINK}/gpp/BUILD/CONFIG directory.This enables users to understand which files are included in a specific configuration.

1.5.2. ARCH

Previous version of DSPLink used to have DSP & HAL specific code distributed throughout the LDRV module logic, which makes the porting of DSPLink a bit difficult job. Now all, DSP & HAL specific code is placed under one directory called ARCH. Logic from this module is exposed through a set of APIs called DSP APIs (such as DSP_init, DSP_start, DSP_stop). This APIs are called by LDRV logic directly. This APIs internally executes logic function by calling DSP interface and HAL functions of the required DSP (indicated by DSP Identifier)

1.5.3. LDRV

LDRV logic is updated to be independent of DSP & HAL code, also device specific implementation of some of the modules like MSGQ, CHNL are now made generic. For platform with shared memory a file zcpy_mqt.c contains logic for zero copy shared memory driver , while dcpy_mqt.c contains logic for DMA copy driver. All these implementation are pluggable and selected by the dynamic configuration

1.6. DSP side sources

1.6.1. File and Directories names

File and directories names are changed. Device specific directories are now renamed with DSP name of the SoC. For example, if the platform is DaVinci then DSP name is DM6446GEM, here DM6446 is SoC number and GEM is the DSP name.

1.6.2. Includes

Directory structure of include directory has been changed. now it contains a new directory C64XX which contains a file c64xxdefs.h. This file defines all common macros which apply to DSP devices of C64+ types. while device specific platform.h file includes this file.

files zcpy_mqt.h, zcpy_data.h, sma_pool.h and dma_pool.h are moved from device specific directories to root of include directory. Since the logic inside these files are generic.

1.6.3. DSP/BIOS™ Configuration files

DSPLink base tci files are now updated to take few arguments from the command line. These arguments are as follows:

- Processor Identifier - all base tci files expects an argument when tconf tool is invoked. this argument is treated as processor
- Location of generated files - now a new command line argument is specified to tconf tool to generate all BIOS generated files at preferred location. This option is -Dconfig.ProgramName.

1.6.4. POOL

Now there two different type of POOL implementation available, SMAPOOL is used on the platform where shared memory is available and DMAPOOL for the platform with PCI or VLYNQ interface.

1.6.5. Samples

There is no change in the logical part of samples, but there are changes in the makesystem part. SOURCES files now refer to the generated BIOS files which are now present in the $DSPLINK/dsp/BUILD/<sample name>/BIOS directory.

1.7. Build System

1.7.1. DSP Files and Directories

For multi-DSP DSPLink, every DSP is identified with Processor Identifier. This identifier is suffixed to DSP name, So the intermediate binaries, libraries and code are generated, reflects the DSP and Processor Identifier for which they are built. For example, if DSP side code is compiled for DM6446GEM device, then

```
<DSPLINK>/dsp/BUILD/...
```

looks like below:

```
<DSPLINK>/dsp/BUILD/DM6446GEM_0/...
```

even the export directory has these changes, so it looks like below:

```
<DSPLINK>/dsp/export/BIN/DspBios/DAVINCI/DM6446GEM_0/...
```

1.7.2. GPP Files and Directories

GPP side BUILD and export directories remains unchanged. Only difference, that previously while building GPP side code, CFG_XXX.c files were copied inside <DSPLINK>/gpp/src/api directory. Now they are copied inside <DSPLINK>/config/BUILD/ directory, this gives the adavantage of not touching the source tree. The <DSPLINK>/config/BUILD driectory is not removed when a user does

```
make  clean clobber
```

it is removed when invoke the static configuration script again.

1.7.3. DSP Side Makesystem

Makesystem in general is revamped to support multi-DSP. The changes done for DSP side are as follows:

- For backward compatibility or for single DSP configuration, when user invokes

  ```
  make
  ```

  command, the default DSP (i.e. DSP with procId = 0) is built.

- For multi-DSP, a multimake script is generated, which on invokation builds binaries and libraries for all DSP configured. It is generated every time when static configuration script is executed. This script is generated under

  ```
  <DSPLINK>/etc/host/scipt/<GPPOS>/
  ```

  directory.

- DSP/BIOS generated files (by tconf/configuro tool) are generated inside BUILD directory, eariler it was generated inside the samples directories. For example, if

DSP message sample is compiled for DM6446GEM device, then BIOS generated files are in:

```
<DSPLINK>/dsp/BUILD/DM6446GEM_0/MESSAGE/BIOS_DEB/
```

for debug build mode.

```
<DSPLINK>/dsp/BUILD/DM6446GEM_0/MESSAGE/BIOS_REL/
```

for release build mode. This driectory is deleted every time whenever sample is recompiled.

1.8. History

- V.01 FEB 12, 2009 Sachin Kumar

  Added revision history and information regarding run SWI samples