EDMA3 Prioritization and Debugging

Henry Yiu

ABSTRACT

The EDMA3 is a high-performance, multi-channel module for the TMS320C64x+ DSP that allows you to program a wide variety of transfer geometries and sequences. In addition, it interacts with the switch central resource (SCR) fabric to compete for bus and read/write access resources. The EDMA3 prioritization scheme is summarized and various program registers are introduced in this application report. Additionally, how to use the EDMA3 debug capability to view its internal operation is demonstrated on a test program running on the TMS320C6455 DSP starter kit (DSK).

Project collateral and source code discussed in this application report can be downloaded from the following URL: http://www-s.ti.com/sc/techlit/spraaq1.zip

Contents

List of Figures

List of Tables

1 EDMA3 Introduction

The EDMA3 controller consists of two major blocks, the EDMA3 channel controller (EDMA3CC) and the EDMA3 transfer controllers (EDMA3TC). The EDMA3CC serves as the user interface, to prioritize incoming software requests or events from peripherals, and submits transfer requests (TRs) to the transfer controllers. The transfer controllers are slaves to the channel controller responsible for data movement; they issue read/write commands to the source and destination addresses programmed for a given transfer.

There should be only one channel controller per device, but there could be several transfer controllers. For example, the C6455 device has four transfer controllers and the DM6446 device has only two. The number of event queues corresponds to the number of transfer controllers. Event queue 0 submits requests to transfer controller 0 and event queue 3 submit requests to transfer controller 3, etc. Figure 1 and Figure 2 shows the transfer controller and channel controller block diagrams.

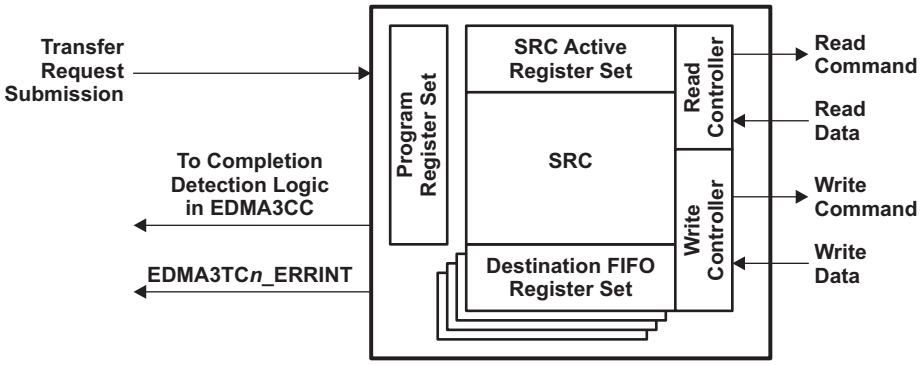

Figure 1. Transfer Controller Block Diagram

2 EDMA3 Prioritization

Figure 2 shows the prioritization diagram from the *TMS320C645x DSP Enhanced DMA (EDMA3) Controller User's Guide* (SPRU966).

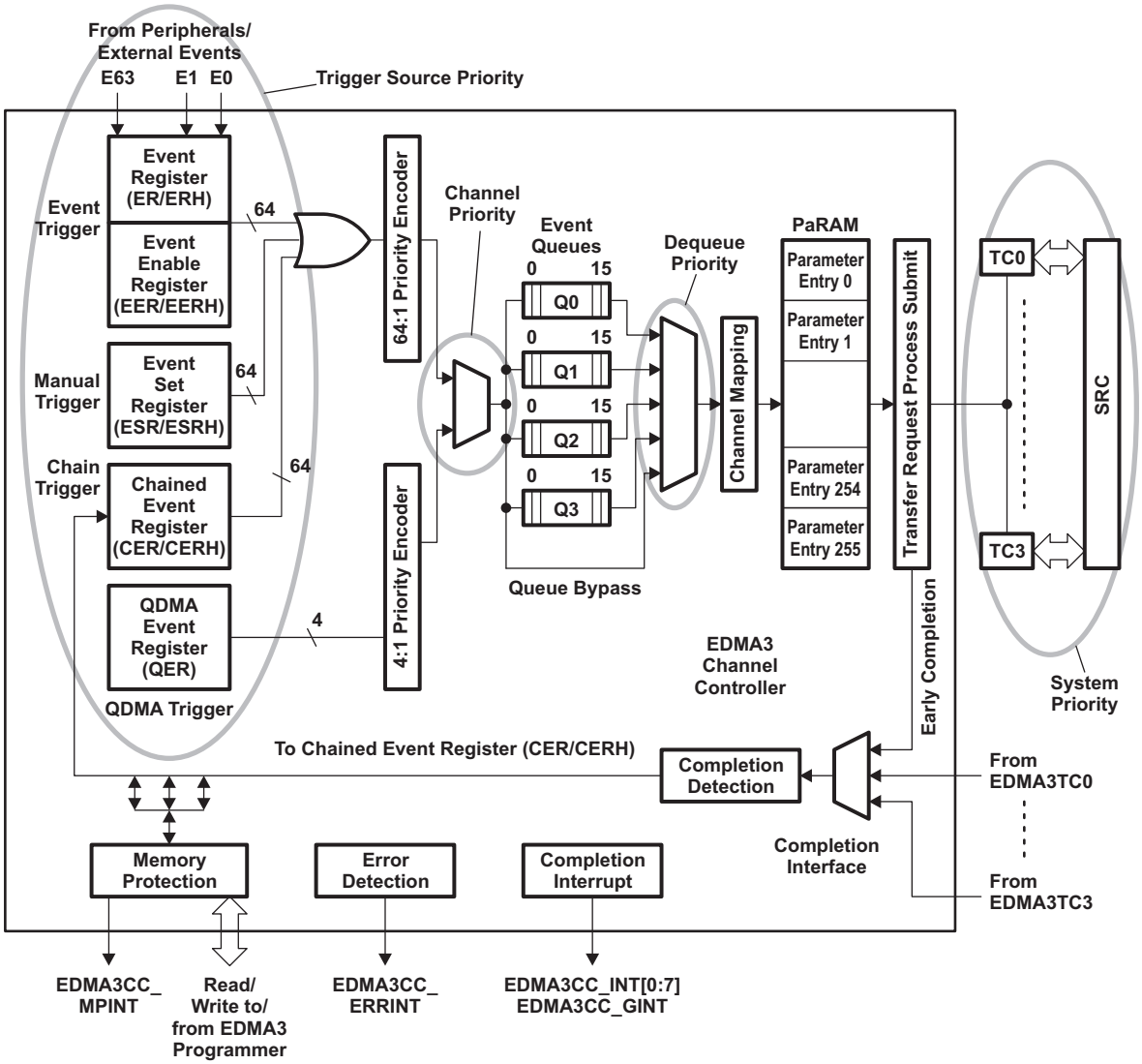

Figure 2. Channel Controller Block Diagram and EDMA3 Prioritization

As mentioned in the *TMS320C645x DSP Enhanced DMA (EDMA3) Controller User's Guide* (SPRU966), EDMA3 prioritization is based on the following priorities:

- Channel Priority - Used when events arrive simultaneously and they need to be submitted to the four queues.
- Trigger Source Priority - Used when one DMA channel is associated with more than one trigger source.
- Dequeue Priority - Queue 0 has the highest priority and queue 3 has the lowest priority. Which queue is used is based on what is programmed into the DMAQNUMn and QDMAQNUM registers.
- Transfer Controller Priority - Every transfer controller has a programmed system priority that is implemented when multiple masters in the system are going for the same endpoint. This priority comes into play at the SCR when several masters are submitting requests to the main SCR. The priority setting ranges from 0 to 7, where 0 is the highest and 7 the lowest. The priority setting is in conjunction with other bus masters on the device.

3 Switched Central Resource (SCR) Bus Priorities

The transfer controller priorities are also the SCR priorities and control how each transfer request competes with other bus masters in the same device. Table 1 shows the system priority setting for the DM6446 device.

Table 1. DM6446 Default Bus Master Priorities

Priority Bit Field	Bus Master	Default Priority Level
VPSSP	VPSS	0 VPSS PCR Register
EDMATC0P	EDMATC0	0 EDMACC QUEPRI Register
EDMATC1P	EDMATC1	0 EDMACC QUEPRI Register
ARM_DMAP	ARM (DMA)	1 MSTPRI0 Register
ARM_CFGP	ARM (CFG)	1 MSTPRI0 Register
C64x+_DMAP	C64+ (DMA)	7 C64x+ MDMAARBE.PRI Register Bit Field
C64x+_CFGP	C64+ (CFG)	1 MSTPRI0 Register
EMACP	EMAC	4 MSTPRI1 Register
USBP	USB	4 MSTPRI1 Register
ATAP	ATA/CF	4 MSTPRI1 Register
VLYNQP	VLYNQ	4 MSTPRI1 Register
VICPP	VICP	5 MSTPRI0 Register

The C6455 bus master priorities are summarized in Table 2; however, more information regarding these priorities can be found in the device-specific user's guides and data sheets.

Table 2. C6455 Default Bus Master Priorities

Priority Bit Field	Bus Master	Default Priority Level
EDMA3 TC0	0 QUEPRI.PRIQ0	EDMA3 UG
EDMA3 TC1	0 QUEPRI.PRIQ1	EDMA3 UG
EDMA3 TC2	0 QUEPRI.PRIQ2	EDMA3 UG
EDMA3 TC3	0 QUEPRI.PRIQ3	EDMA3 UG
SRIO	1 PRI_ALLOC.SRIO	Datasheet
EMAC	1 PRI_ALLOC.EMAC	Datasheet
HPI/PCI	2 PRI_ALLOC.HOST	Datasheet
Megamodule Master	7 MDMAARB.PRI	Megamodule UG

The master megamodule uses the master DMA (MDMA) of the megamodule's extended memory controller (EMC). The master DMA is typically used for CPU/cache accesses to memory beyond the L2 level. These accesses are in the form of cache line allocates, writebacks, and non-cacheable loads and stores to system memory.

There is also a megamodule slave that uses the slave DMA (SDMA) of the megamodule's EMC. The slave DMA provides access from resources inside the megamodule, to the system masters found outside the megamodule, such as DMA controllers, HPI, etc.

Table 3 summarizes the priority level within the megamodule.

Table 3. C6455 Megamodule Priorities

Priority Bit Field	Bus Master	Default Priority Level
CPU to L1D ,L2, EMC	1 CPUARB.PRI, 16 CPUARB.MAXWAIT	Megamodule UG
IDMA channel 0	0 fixed	Megamodule UG
IDMA channel 1	0 IDMA1_COUNT.PRI, 32 IDMAARB.MAXWAIT	Megamodule UG
User Cohere (Global)	0 fixed	Megamodule UG
User Cohere (Block)	7 fixed, 32 UCARB.MAXWAIT	Megamodule UG
Megamodule Slave	X defined by master, 1 SDMAARB.MAXWAIT	Megamodule UG

4 Testing EDMA3 Debug and Prioritization Features

Based on the priority settings in the previous section, the dequeue and transfer controller priorities have the most important and frequent impact to the system's performance. The channel priority and trigger source priority should be short and can get resolved quickly.

A test program has been written to find the priorities associated with the EDMA3 and SCR. This program is modified based on the dsk_app project that comes with the C6455 DSK. The important features are listed below:

- In the `blinkLED` function, the CSL functions are added to print out the staus of the four queues, especially the watermark values of each queue. The watermark tracks the entries that have been in queue n since reset or since the last time of the watermark. The function then immediately clears the queue watermark back to zero. Since the LED blinks every second, the function prints out the watermark every second this program is running.
- The load function is modified. Instead of adding a CPU loading, it adds another EDMA channel so that it continuously copies a memory region from the `gLoadSrcAddress` to the `gLoadDstAddress` location.
- In the `edmaInit` function, the `QUEPRI` register is modified so that TC0 has a highier priority than TC1, TC1 has a higher priority than TC2, and so on. This can be modified to test out various conditions.

You can use this program to modify various parameters of the EDMA channels and test the effects with respect to the EDMA channels.

5 References

- *TMS320C645x DSP Enhanced DMA (EDMA3) Controller User's Guide* (SPRU966)
- *TMS320DM644x DMSoC Enhanced Direct Memory Access (EDMA) Controller User's Guide* (SPRUE23)
- *TMS320C6455 Fixed-Point Digital Signal Processor* (SPRS276)
- *TMS320DM6446 Digital Media System-on-Chip* (SPRS283)

Appendix A C6455 DSK Test Code for EDMA3

A.1 C6455 DSK Test Code

The following code was modified from the DSK example.

```
/*
 *  dsk_app.c (version 1.00)
 *

#include "dsk_appcfg.h"

#include "dsk6455.h"
#include "dsk6455_led.h"
#include "dsk6455_dip.h"
#include "dsk6455_aic23.h"

#include <stdio.h>
#include <csl.h>
#include <csl_edma3.h>
#include <csl_intc.h>
#include <csl_cache.h>
#include <soc.h>
#include <hwi.h>

#define BUFFSIZE 2048
#define PING     0
#define PONG     1

#define LOADBUFFSIZE  128
#define EDMACHAN0   3
#define EDMACHAN1   4
#define EDMACHAN2   6

//#pragma DATA_SECTION (gAvailableChannel, "mem_ext")
//#pragma DATA_ALIGN (gAvailableChannel, 128)
//Int32 gAvailableChannel[MAX_CHAN] = {3, 4, 5, 6, 7, 8, 9, 10, 11, 18, 19, 21,
//                                     22, 23, 24, 25, 26, 27, 33, 34, 35, 36,
//                     37, 38, 39, 41, 42, 43, 46, 47};

#pragma DATA_SECTION (gLoadSrcAddress, "mem_ext")
#pragma DATA_ALIGN (gLoadSrcAddress, 128)
Int32 gLoadSrcAddress[LOADBUFFSIZE] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};

#pragma DATA_SECTION (gLoadDstAddress, "mem_ext")
#pragma DATA_ALIGN (gLoadDstAddress, 128)
Int32 gLoadDstAddress[LOADBUFFSIZE] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

#pragma DATA_SECTION (gBufferRcvPing, "mem_ext")
#pragma DATA_ALIGN (gBufferRcvPing, 128)
Int16 gBufferRcvPing[BUFFSIZE]; // Receive PING buffer

#pragma DATA_SECTION (gBufferRcvPong, "mem_ext")
#pragma DATA_ALIGN (gBufferRcvPong, 128)
Int16 gBufferRcvPong[BUFFSIZE]; // Receive PONG buffer

#pragma DATA_SECTION (gBufferXmtPing, "mem_ext")
#pragma DATA_ALIGN (gBufferXmtPing, 128)
Int16 gBufferXmtPing[BUFFSIZE]; // Transmit PING buffer

#pragma DATA_SECTION (gBufferXmtPong, "mem_ext")
#pragma DATA_ALIGN (gBufferXmtPong, 128)
Int16 gBufferXmtPong[BUFFSIZE]; // Transmit PONG buffer

/* Codec configuration settings */
DSK6455_AIC23_Config config = {
    0x0017, // 0 DSK6455_AIC23_LEFTINVOL  Left line input channel volume
    0x0017, // 1 DSK6455_AIC23_RIGHTINVOL Right line input channel volume
    0x01f7, // 2 DSK6455_AIC23_LEFTHPVOL  Left channel headphone volume
    0x01f7, // 3 DSK6455_AIC23_RIGHTHPVOL Right channel headphone volume
    0x0010, // 4 DSK6455_AIC23_ANAPATH    Analog audio path control
```

```
    0x0000, // 5 DSK6455_AIC23_DIGPATH    Digital audio path control
    0x0000, // 6 DSK6455_AIC23_POWERDOWN  Power down control
    0x0043, // 7 DSK6455_AIC23_DIGIF      Digital audio interface format
    0x0081, // 8 DSK6455_AIC23_SAMPLERATE Sample rate control
    0x0001  // 9 DSK6455_AIC23_DIGACT     Digital interface activation
};

/* Intc declaration */
CSL_IntcContext                 intcContext;
CSL_IntcEventHandlerRecord      EventHandler[100];
CSL_IntcObj                     intcObjEdma;
CSL_IntcHandle                  hIntcEdma;
CSL_IntcGlobalEnableState       state;
CSL_IntcEventHandlerRecord      EventRecord;
CSL_IntcParam                   vectId;
CSL_Edma3HwDmaChannelSetup      dmahwSetup[CSL_EDMA3_NUM_DMACH] =
CSL_EDMA3_DMACHANNELSETUP_DEFAULT;
CSL_Edma3HwSetup                hwSetup = {&dmahwSetup[0],NULL};

/* Globals */

/* Edma module */
CSL_Edma3Handle     hModule;
CSL_Edma3Obj        edmaObj;

CSL_Edma3ChannelObj        chObjLoad0, chObjLoad1, chObjLoad2;
CSL_Edma3ChannelHandle     hChannelLoad0, hChannelLoad1, hChannelLoad2;

//*****************************************************************************
// Loading Channel Parameter RAM setup
//*****************************************************************************

/* PaRAM settings for sets 69 (loading) and 68 (loading reload) */
CSL_Edma3ParamSetup gParamSetupLoad0 = {
    CSL_EDMA3_OPT_MAKE                      // option -      OPT
         (CSL_EDMA3_ITCCH_DIS,     \
        CSL_EDMA3_TCCH_DIS,       \
        CSL_EDMA3_ITCINT_DIS,     \
        CSL_EDMA3_TCINT_EN,       \
        0, CSL_EDMA3_TCC_NORMAL,  \
        CSL_EDMA3_FIFOWIDTH_NONE, \
        CSL_EDMA3_STATIC_DIS,     \
        CSL_EDMA3_SYNC_AB,        \
        CSL_EDMA3_ADDRMODE_INCR,  \
        CSL_EDMA3_ADDRMODE_INCR),
    (Uint32)&gLoadSrcAddress,               // srcAddr -     SRC
    CSL_EDMA3_CNT_MAKE(4, LOADBUFFSIZE),    // aCntbCnt -    (ACNT, BCNT)
    (Uint32)&gLoadDstAddress,               // dstAddr -     DST
    CSL_EDMA3_BIDX_MAKE(4, 4),              // srcDstBidx -  (SRCBIDX, DSTBIDX)
    CSL_EDMA3_LINKBCNTRLD_MAKE(0x4000+32*68, 1),  // linkBcntrld - (LINK, BCNTRLD)
    CSL_EDMA3_CIDX_MAKE(0,0),               // srcDstCidx -  (SRCCIDX, DSTCIDX)
    1                                       // cCnt -        CCNT
};

/* PaRAM settings for sets 71 (loading) and 70 (loading reload) */
CSL_Edma3ParamSetup gParamSetupLoad1 = {
    CSL_EDMA3_OPT_MAKE                      // option -      OPT
        (CSL_EDMA3_ITCCH_DIS,     \
        CSL_EDMA3_TCCH_DIS,       \
        CSL_EDMA3_ITCINT_DIS,     \
        CSL_EDMA3_TCINT_EN,       \
        0, CSL_EDMA3_TCC_NORMAL,  \
        CSL_EDMA3_FIFOWIDTH_NONE, \
        CSL_EDMA3_STATIC_DIS,     \
        CSL_EDMA3_SYNC_AB,        \
        CSL_EDMA3_ADDRMODE_INCR,  \
        CSL_EDMA3_ADDRMODE_INCR),
    (Uint32)&gLoadSrcAddress,               // srcAddr -     SRC
    CSL_EDMA3_CNT_MAKE(4, LOADBUFFSIZE),    // aCntbCnt -    (ACNT, BCNT)
    (Uint32)&gLoadDstAddress,               // dstAddr -     DST
    CSL_EDMA3_BIDX_MAKE(4, 4),              // srcDstBidx -  (SRCBIDX, DSTBIDX)
    CSL_EDMA3_LINKBCNTRLD_MAKE(0x4000+32*70, 1),  // linkBcntrld - (LINK, BCNTRLD)
    CSL_EDMA3_CIDX_MAKE(0,0),               // srcDstCidx -  (SRCCIDX, DSTCIDX)
    1                                       // cCnt -        CCNT
```

```
};

/* PaRAM settings for sets 73 (loading) and 72 (loading reload) */
CSL_Edma3ParamSetup gParamSetupLoad2 = {
    CSL_EDMA3_OPT_MAKE                      // option -      OPT
        (CSL_EDMA3_ITCCH_DIS,     \
        CSL_EDMA3_TCCH_DIS,       \
        CSL_EDMA3_ITCINT_DIS,     \
        CSL_EDMA3_TCINT_EN,       \
        0, CSL_EDMA3_TCC_NORMAL,  \
        CSL_EDMA3_FIFOWIDTH_NONE, \
        CSL_EDMA3_STATIC_DIS,     \
        CSL_EDMA3_SYNC_AB,        \
        CSL_EDMA3_ADDRMODE_INCR,  \
        CSL_EDMA3_ADDRMODE_INCR),
    (Uint32)&gLoadSrcAddress,               // srcAddr -     SRC
    CSL_EDMA3_CNT_MAKE(4, LOADBUFFSIZE),    // aCntbCnt -    (ACNT, BCNT)
    (Uint32)&gLoadDstAddress,               // dstAddr -     DST
    CSL_EDMA3_BIDX_MAKE(4, 4),              // srcDstBidx -  (SRCBIDX, DSTBIDX)
    CSL_EDMA3_LINKBCNTRLD_MAKE(0x4000+32*72, 1),  // linkBcntrld - (LINK, BCNTRLD)
    CSL_EDMA3_CIDX_MAKE(0,0),               // srcDstCidx -  (SRCCIDX, DSTCIDX)
    1                                       // cCnt -        CCNT
};

//*************************************************************************
// McBSP Used Parameter RAM setup
//*************************************************************************

/* PaRAM settings for sets 12 (transmit) and 67 (transmit ping reload) */
CSL_Edma3ParamSetup gParamSetupXmtPing = {  // PaRAM Set Structure for transmit ping buffer
    CSL_EDMA3_OPT_MAKE                      // option -      OPT
        (CSL_EDMA3_ITCCH_DIS,     \
        CSL_EDMA3_TCCH_DIS,       \
        CSL_EDMA3_ITCINT_DIS,     \
        CSL_EDMA3_TCINT_EN,       \
        14, CSL_EDMA3_TCC_NORMAL, \
        CSL_EDMA3_FIFOWIDTH_NONE, \
        CSL_EDMA3_STATIC_DIS,     \
        CSL_EDMA3_SYNC_A,         \
        CSL_EDMA3_ADDRMODE_INCR,  \
        CSL_EDMA3_ADDRMODE_INCR),
    (Uint32)&gBufferXmtPing,                // srcAddr -     SRC
    CSL_EDMA3_CNT_MAKE(2,BUFFSIZE),         // aCntbCnt -    (ACNT, BCNT)
    (Uint32)0x02900004,                     // dstAddr -     DST
    CSL_EDMA3_BIDX_MAKE(2,0),               // srcDstBidx -  (SRCBIDX, DSTBIDX)
    CSL_EDMA3_LINKBCNTRLD_MAKE(0x4000+32*66, 1),  // linkBcntrld - (LINK, BCNTRLD)
    CSL_EDMA3_CIDX_MAKE(0,0),               // srcDstCidx -  (SRCCIDX, DSTCIDX)
    1                                       // cCnt -        CCNT
};

/* PaRAM settings for set 66 (transmit pong reload) */
CSL_Edma3ParamSetup gParamSetupXmtPong = {  // PaRAM Set Structure for transmit pong buffer
    CSL_EDMA3_OPT_MAKE                      // option -      OPT
        (CSL_EDMA3_ITCCH_DIS,     \
        CSL_EDMA3_TCCH_DIS,       \
        CSL_EDMA3_ITCINT_DIS,     \
        CSL_EDMA3_TCINT_EN,       \
        14, CSL_EDMA3_TCC_NORMAL, \
        CSL_EDMA3_FIFOWIDTH_NONE, \
        CSL_EDMA3_STATIC_DIS,     \
        CSL_EDMA3_SYNC_A,         \
        CSL_EDMA3_ADDRMODE_INCR,  \
        CSL_EDMA3_ADDRMODE_INCR),
    (Uint32)gBufferXmtPong,                 // srcAddr -     SRC
    CSL_EDMA3_CNT_MAKE(2,BUFFSIZE),         // aCntbCnt -    (ACNT, BCNT)
    (Uint32)0x02900004,                     // dstAddr -     DST
    CSL_EDMA3_BIDX_MAKE(2,0),               // srcDstBidx -  (SRCBIDX, DSTBIDX)
    CSL_EDMA3_LINKBCNTRLD_MAKE(0x4000+32*67, 1),  // linkBcntrld - (LINK, BCNTRLD)
    CSL_EDMA3_CIDX_MAKE(0,0),               // srcDstCidx -  (SRCCIDX, DSTCIDX)
    1                                       // cCnt -        CCNT
};

/* PaRAM settings for sets 13 (receive) and 65 (receive ping reload) */
CSL_Edma3ParamSetup gParamSetupRcvPing = {  // PaRAM Set Structure for receive ping buffer
```

```
    CSL_EDMA3_OPT_MAKE                      // option -      OPT
        (CSL_EDMA3_ITCCH_DIS,     \
        CSL_EDMA3_TCCH_DIS,       \
        CSL_EDMA3_ITCINT_DIS,     \
        CSL_EDMA3_TCINT_EN,       \
        15, CSL_EDMA3_TCC_NORMAL, \
        CSL_EDMA3_FIFOWIDTH_NONE, \
        CSL_EDMA3_STATIC_DIS,     \
        CSL_EDMA3_SYNC_A,         \
        CSL_EDMA3_ADDRMODE_INCR,  \
        CSL_EDMA3_ADDRMODE_INCR),
    (Uint32)0x02900000,                     // srcAddr -     SRC
    CSL_EDMA3_CNT_MAKE(2,BUFFSIZE),         // aCntbCnt -    (ACNT, BCNT)
    (Uint32)&gBufferRcvPing,                // dstAddr -     DST
    CSL_EDMA3_BIDX_MAKE(0,2),               // srcDstBidx -  (SRCBIDX, DSTBIDX)
    CSL_EDMA3_LINKBCNTRLD_MAKE(0x4000+32*64, 1),  // linkBcntrld - (LINK, BCNTRLD)
    CSL_EDMA3_CIDX_MAKE(0,0),               // srcDstCidx -  (SRCCIDX, DSTCIDX)
    1                                       // cCnt -        CCNT
};

/* PaRAM settings for set 64 (transmit pong reload) */
CSL_Edma3ParamSetup gParamSetupRcvPong = {  // PaRAM Set Structure for receive pong buffer
    CSL_EDMA3_OPT_MAKE                      // option -      OPT
        (CSL_EDMA3_ITCCH_DIS,     \
        CSL_EDMA3_TCCH_DIS,       \
        CSL_EDMA3_ITCINT_DIS,     \
        CSL_EDMA3_TCINT_EN,       \
        15, CSL_EDMA3_TCC_NORMAL, \
        CSL_EDMA3_FIFOWIDTH_NONE, \
        CSL_EDMA3_STATIC_DIS,     \
        CSL_EDMA3_SYNC_A,         \
        CSL_EDMA3_ADDRMODE_INCR,  \
        CSL_EDMA3_ADDRMODE_INCR),
    (Uint32)0x02900000,                     // srcAddr -     SRC
    CSL_EDMA3_CNT_MAKE(2,BUFFSIZE),         // aCntbCnt -    (ACNT, BCNT)
    (Uint32)&gBufferRcvPong,                // dstAddr -     DST
    CSL_EDMA3_BIDX_MAKE(0,2),               // srcDstBidx -  (SRCBIDX, DSTBIDX)
    CSL_EDMA3_LINKBCNTRLD_MAKE(0x4000+32*65, 1),  // linkBcntrld - (LINK, BCNTRLD)
    CSL_EDMA3_CIDX_MAKE(0,0),               // srcDstCidx -  (SRCCIDX, DSTCIDX)
    1                                       // cCnt -        CCNT
};

/*
 *  copyData() - Copy one buffer with length elements to another.
 */
void copyData(Int16 *inbuf, Int16 *outbuf, Int16 length)
{
    Int16 i = 0;

    for (i = 0; i < length; i++) {
            outbuf[i] = inbuf[i];
    }
}

/* ---------------------- Interrupt Service Routine -------------------- */

/*
 *  edmaHwi() - Interrupt service routine for the DMA transfer.  It is
 *              triggered when a complete DMA receive frame has been
 *              transferred.   The edmaHwi ISR is inserted into the interrupt
 *              vector table at compile time through a setting in the DSP/BIOS
 *              configuration under Scheduling --> HWI --> HWI_INT8.  edmaHwi
 *              uses the DSP/BIOS Dispatcher to save register state and make
 *              sure the ISR co-exists with other DSP/BIOS functions.
 */
void edmaHwi(void)
{
       Uint32 intr;
    static Uint32       pingOrPong = PING;
       static Int16        xmtdone = 0, rcvdone = 0, loaddone = 0;

       /* Check for pending EDMA event interrupts (IPR) */
       intr = *((Uint32*)0x02a01068);
```

```
            if (intr & (0x1 << EDMACHAN0))
            {
                loaddone = 1;
            }
            if (intr & (0x1 << EDMACHAN1))
            {
                loaddone = 1;
            }
            if (intr & (0x1 << EDMACHAN2))
            {
                loaddone = 1;
            }
        if (intr & 0x4000)
            {
                xmtdone = 1;
            }
            if (intr & 0x8000)
            {
                rcvdone = 1;
            }
            if (xmtdone && rcvdone)
            {
                if (pingOrPong == PING)
                        {
                    pingOrPong = PONG;
//                  copyData(gBufferRcvPing, gBufferXmtPing, BUFFSIZE);
                                SWI_or(&processBufferSwi, PING);
                    }
                    else
                    {
                        pingOrPong = PING;
//                      copyData(gBufferRcvPong, gBufferXmtPong, BUFFSIZE);
                                SWI_or(&processBufferSwi, PONG);
                    }
                    rcvdone = 0;
                    xmtdone = 0;
            }

        if (loaddone)
            {
                  loaddone = 0;
            }

        /* Clear CPU interrupt event 72 (EVTCLR2) */
            *((Uint32*)0x1800048) = 0x00000100;

            /* Clear processed EDMA event interrupts (ICR) */
            *((Uint32*)0x02a01070) = intr;
    }


    /* ------------------------------ Threads ------------------------------ */

    /*
     *  processBuffer() - Process audio data once it has been received.
     */
    void processBuffer(void)
    {
        Uint32 pingPong;

        /* Get contents of mailbox posted by edmaHwi */
        pingPong =  SWI_getmbox();

        /* Copy data from transmit to receive, could process audio here */
        if (pingPong == PING) {
            /* Toggle + #3 as a visual cue */
            DSK6455_LED_toggle(3);

            /* Copy receive PING buffer to transmit PING buffer */
                        CACHE_invL1d (gBufferRcvPing, BUFFSIZE*2, CACHE_WAIT);
            copyData(gBufferRcvPing, gBufferXmtPing, BUFFSIZE);
                        CACHE_wbL1d (gBufferXmtPing, BUFFSIZE*2, CACHE_WAIT);

        } else {
```

```
            /* Toggle LED #2 as a visual cue */
            DSK6455_LED_toggle(2);

            /* Copy receive PONG buffer to transmit PONG buffer */
                        CACHE_invL1d (gBufferRcvPong, BUFFSIZE*2, CACHE_WAIT);
            copyData(gBufferRcvPong, gBufferXmtPong, BUFFSIZE);
                        CACHE_wbL1d (gBufferXmtPong, BUFFSIZE*2, CACHE_WAIT);
        }
    }

    /*
     *  blinkLED() - Periodic thread (PRD) that toggles LED #0 every 500ms if
     *               DIP switch #0 is depressed.  The thread is configured
     *               in the DSP/BIOS configuration tool under Scheduling -->
     *               PRD --> PRD_blinkLed.  The period is set there at 500
     *               ticks, with each tick corresponding to 1ms in real
     *               time.
     */
    void blinkLED(void)
    {
            CSL_Edma3QueStat                                    info;

    //      int record[64];
    //      for (i = 0; i < 4; i++)
    //                  record[i] = *( (Uint32*)(0x02A00600 + i) );

            info.que = CSL_EDMA3_QUE_0;
            CSL_edma3GetHwStatus (hModule ,CSL_EDMA3_QUERY_QUESTATUS, &info);
            LOG_printf (&trace, "Queue = %d   WM = %d", info.que, info.waterMark);

            info.que = CSL_EDMA3_QUE_1;
            CSL_edma3GetHwStatus (hModule ,CSL_EDMA3_QUERY_QUESTATUS, &info);
            LOG_printf (&trace, "Queue = %d   WM = %d", info.que, info.waterMark);

            info.que = CSL_EDMA3_QUE_2;
            CSL_edma3GetHwStatus (hModule ,CSL_EDMA3_QUERY_QUESTATUS, &info);
            LOG_printf (&trace, "Queue = %d   WM = %d", info.que, info.waterMark);

            info.que = CSL_EDMA3_QUE_3;
            CSL_edma3GetHwStatus (hModule ,CSL_EDMA3_QUERY_QUESTATUS, &info);
            LOG_printf (&trace, "Queue = %d   WM = %d", info.que, info.waterMark);

            hModule->regs->CCERRCLR = CSL_FMKT(EDMA3CC_CCERRCLR_QTHRXCD0, CLEAR);
            hModule->regs->CCERRCLR = CSL_FMKT(EDMA3CC_CCERRCLR_QTHRXCD1, CLEAR);
            hModule->regs->CCERRCLR = CSL_FMKT(EDMA3CC_CCERRCLR_QTHRXCD2, CLEAR);
            hModule->regs->CCERRCLR = CSL_FMKT(EDMA3CC_CCERRCLR_QTHRXCD3, CLEAR);

        /* Toggle LED #0 if DIP switch #0 is off (depressed) */
        if (!DSK6455_DIP_get(0))
           {
            DSK6455_LED_toggle(0);
            }
    }

    /*
     *  load() - PRD that simulates a roughly 20% dummy CPU load if
     *           DIP switch #1 is depressed.  The thread is configured in
     *           the DSP/BIOS configuration tool under Scheduling --> PRD
     *           PRD_load.  The period is set there at 10 ticks, which each tick
     *           corresponding to 1ms in real time.
     */
    void load(void)
    {
        volatile Uint32 i, j, kkk;

        if (!DSK6455_DIP_get(1))
           {
                    for (i = 0; i < 1; i++)
                    {
                                /* Manually trigger the loading channel */
    //      CSL_edma3HwChannelControl(hChannelLoad, CSL_EDMA3_CMD_CHANNEL_SET, NULL);
                                (hModule->regs->ESR) |= (0x1 << EDMACHAN0);
                                // Wait for submission to transfer controller.
    //                          while (((hModule->regs->ESR) >> EDMACHAN0) & 0x1);
```

```
                                (hModule->regs->ESR) |= (0x1 << EDMACHAN1);
                                // Wait for submission to transfer controller.
//                              while (((hModule->regs->ESR) >> EDMACHAN1) & 0x1);

                                (hModule->regs->ESR) |= (0x1 << EDMACHAN2);

                                // Wait for submission to transfer controller.
//                              while (((hModule->regs->ESR) >> EDMACHAN2) & 0x1);

                                // Check for missed event and clear if necessary.
//                              if (((hModule->regs->EMR) >> EDMACHAN0) & 0x1) {
//                              LOG_printf (&trace, "DEBUG: EDMA event missed.");
//                              (hModule->regs->EMCR) = (1 << EDMACHAN0);
//                              }
                }
     }
}

/*
 *  main() - Main code routine, initializes codec and starts EMDA transfer
 */

void edmaInit()
{
    Uint32                      i, reg, regh;
    CSL_Edma3ParamHandle        hParamBasic;
    CSL_Edma3ParamSetup         ParamLoad;
    CSL_Edma3ChannelObj         chObjXmt, chObjRcv;
    CSL_Edma3CmdIntr            regionIntr;
    CSL_Edma3CmdDrae            regionAccess;
    CSL_Edma3CmdQuePri          queuePriSetting;
    CSL_Edma3ChannelHandle      hChannelXmt, hChannelRcv;
    CSL_Edma3Context            context;
    CSL_Edma3ChannelAttr        chAttrXmt, chAttrRcv, chAttrLoad;
    CSL_Status                  status;

    /* Disable global interrupts */
    HWI_disable();

        /* Install HWI handler */
        HWI_dispatchPlug(4, (Fxn)&edmaHwi, 0, NULL);
    HWI_eventMap(4, 72);      // EDMA3CC_INT1 for (DRAE1: region 1)
        C64_enableIER(0x10);

    /* Module initialization */
    status = CSL_edma3Init(&context);
    if (status != CSL_SOK) {
        printf ("Edma module initialization failed\n");
        return;
    }

    /* Edma module open */
    hModule = CSL_edma3Open(&edmaObj,CSL_EDMA3,NULL,&status);

    /* Setup the DRAE masks
     * DRAE enable(Bits 0-15) for the shadow region 1.
     */
    regionAccess.region = CSL_EDMA3_REGION_1 ;
    regionAccess.drae =   0xFFFFFFFF ;
    regionAccess.draeh =  0xFFFFFFFF ;
    CSL_edma3HwControl(hModule,CSL_EDMA3_CMD_DMAREGION_ENABLE, &regionAccess);

       /* Setup QUEPRI register */
       queuePriSetting.que = CSL_EDMA3_QUE_0;
       queuePriSetting.pri = CSL_EDMA3_QUE_PRI_0;
       CSL_edma3HwControl (hModule,CSL_EDMA3_CMD_QUEPRIORITY_SET, &queuePriSetting);
       queuePriSetting.que = CSL_EDMA3_QUE_1;
       queuePriSetting.pri = CSL_EDMA3_QUE_PRI_1;
       CSL_edma3HwControl (hModule,CSL_EDMA3_CMD_QUEPRIORITY_SET, &queuePriSetting);
       queuePriSetting.que = CSL_EDMA3_QUE_2;
       queuePriSetting.pri = CSL_EDMA3_QUE_PRI_2;
       CSL_edma3HwControl (hModule,CSL_EDMA3_CMD_QUEPRIORITY_SET, &queuePriSetting);
       queuePriSetting.que = CSL_EDMA3_QUE_3;
```

```
        queuePriSetting.pri = CSL_EDMA3_QUE_PRI_3;
        CSL_edma3HwControl (hModule,CSL_EDMA3_CMD_QUEPRIORITY_SET, &queuePriSetting);

    /* DMA chanel --> PaRAM mapping, queue assignment */
//  dmahwSetup[14].paramNum = 12;
//      dmahwSetup[14].que =      CSL_EDMA3_QUE_1;
//      dmahwSetup[15].paramNum = 13;
//      dmahwSetup[15].que =      CSL_EDMA3_QUE_1;
    CSL_edma3HwSetup(hModule,&hwSetup);

        // --- Setup Loading Channel 0 ---

        chAttrLoad.regionNum = CSL_EDMA3_REGION_1;
        chAttrLoad.chaNum = EDMACHAN0;
        hChannelLoad0 = CSL_edma3ChannelOpen(&chObjLoad0, CSL_EDMA3, &chAttrLoad, &status);

    /* Set up parameter set 68 as EDMA reload */
        hParamBasic = CSL_edma3GetParamHandle(hChannelLoad0,68,&status);
        ParamLoad = gParamSetupLoad0;
        ParamLoad.option |= CSL_FMKR(17,12,EDMACHAN0);
    CSL_edma3ParamSetup(hParamBasic,&ParamLoad);

        /* Set up parameter set 69 as EDMA */
        hParamBasic = CSL_edma3GetParamHandle(hChannelLoad0,69,&status);
        ParamLoad = gParamSetupLoad0;
        ParamLoad.option |= CSL_FMKR(17,12,EDMACHAN0);
        CSL_edma3ParamSetup(hParamBasic,&ParamLoad);

        /* Set up channel and queue relationships */
        CSL_edma3HwChannelSetupParam(hChannelLoad0, 69);
        CSL_edma3HwChannelSetupQue(hChannelLoad0, CSL_EDMA3_QUE_1);

        // --- Setup Loading Channel 1 ---

        chAttrLoad.regionNum = CSL_EDMA3_REGION_1;
        chAttrLoad.chaNum = EDMACHAN1;
        hChannelLoad1 = CSL_edma3ChannelOpen(&chObjLoad1, CSL_EDMA3, &chAttrLoad, &status);

    /* Set up parameter set 70 as EDMA reload */
        hParamBasic = CSL_edma3GetParamHandle(hChannelLoad1,70,&status);
        ParamLoad = gParamSetupLoad1;
        ParamLoad.option |= CSL_FMKR(17,12,EDMACHAN1);
    CSL_edma3ParamSetup(hParamBasic,&ParamLoad);

        /* Set up parameter set 71 as EDMA */
        hParamBasic = CSL_edma3GetParamHandle(hChannelLoad1,71,&status);
        ParamLoad = gParamSetupLoad1;
        ParamLoad.option |= CSL_FMKR(17,12,EDMACHAN1);
        CSL_edma3ParamSetup(hParamBasic,&ParamLoad);

        /* Set up channel and queue relationships */
        CSL_edma3HwChannelSetupParam(hChannelLoad1, 71);
        CSL_edma3HwChannelSetupQue(hChannelLoad1, CSL_EDMA3_QUE_1);

        // --- Setup Loading Channel 2 ---

        chAttrLoad.regionNum = CSL_EDMA3_REGION_1;
        chAttrLoad.chaNum = EDMACHAN2;
        hChannelLoad2 = CSL_edma3ChannelOpen(&chObjLoad2, CSL_EDMA3, &chAttrLoad, &status);

    /* Set up parameter set 72 as EDMA reload */
        hParamBasic = CSL_edma3GetParamHandle(hChannelLoad2,72,&status);
        ParamLoad = gParamSetupLoad2;
        ParamLoad.option |= CSL_FMKR(17,12,EDMACHAN2);
    CSL_edma3ParamSetup(hParamBasic,&ParamLoad);

     /* Set up parameter set 73 as EDMA */
        hParamBasic = CSL_edma3GetParamHandle(hChannelLoad2,73,&status);
        ParamLoad = gParamSetupLoad2;
        ParamLoad.option |= CSL_FMKR(17,12,EDMACHAN2);
        CSL_edma3ParamSetup(hParamBasic,&ParamLoad);

        /* Set up channel and queue relationships */
        CSL_edma3HwChannelSetupParam(hChannelLoad2, 73);
```

```
   CSL_edma3HwChannelSetupQue(hChannelLoad2, CSL_EDMA3_QUE_1);

/* --- Setup Transmit Channel (McBSP1 transmit, param 12) --- */

/* Channel open */
chAttrXmt.regionNum = CSL_EDMA3_REGION_1;
chAttrXmt.chaNum = CSL_EDMA3_CHA_XEVT1;             // Channel 14
hChannelXmt = CSL_edma3ChannelOpen(&chObjXmt, CSL_EDMA3, &chAttrXmt, &status);

/* Set up parameter block 67 as EDMA transmit ping reload */
hParamBasic = CSL_edma3GetParamHandle(hChannelXmt,67,&status);
status = CSL_edma3ParamSetup(hParamBasic,&gParamSetupXmtPing);

/* Set up parameter block 66 as EDMA transmit pong reload */
hParamBasic = CSL_edma3GetParamHandle(hChannelXmt,66,&status);
CSL_edma3ParamSetup(hParamBasic,&gParamSetupXmtPong);

/* Set up parameter block 12 as EDMA transmit (start with copy of ping) */
hParamBasic = CSL_edma3GetParamHandle(hChannelXmt,12,&status);
CSL_edma3ParamSetup(hParamBasic,&gParamSetupXmtPing);

/* Set up channel and queue relationships */
CSL_edma3HwChannelSetupParam(hChannelXmt, 12);
CSL_edma3HwChannelSetupQue(hChannelXmt, CSL_EDMA3_QUE_1);   // McBSP can only use TC1.

/* --- Setup Receive Chan vnel (McBSP1 receive, param 13) --- */

/* Channel open */
chAttrRcv.regionNum = CSL_EDMA3_REGION_1;
chAttrRcv.chaNum = CSL_EDMA3_CHA_REVT1;              // Channel 15
hChannelRcv = CSL_edma3ChannelOpen(&chObjRcv, CSL_EDMA3, &chAttrRcv, &status);

/* Set up parameter block 65 as EDMA transmit ping reload */
hParamBasic = CSL_edma3GetParamHandle(hChannelRcv,65,&status);
status = CSL_edma3ParamSetup(hParamBasic,&gParamSetupRcvPing);

/* Set up parameter block 64 as EDMA transmit pong reload */
hParamBasic = CSL_edma3GetParamHandle(hChannelRcv,64,&status);
CSL_edma3ParamSetup(hParamBasic,&gParamSetupRcvPong);

/* Set up parameter block 13 as EDMA transmit (start with copy of ping) */
hParamBasic = CSL_edma3GetParamHandle(hChannelRcv,13,&status);
CSL_edma3ParamSetup(hParamBasic,&gParamSetupRcvPing);

/* Set up channel and queue relationships */
CSL_edma3HwChannelSetupParam(hChannelRcv, 13);
CSL_edma3HwChannelSetupQue(hChannelRcv, CSL_EDMA3_QUE_1);       // McBSP can only use TC1.

/* --- Enable EDMA region interrupts --- */

regionIntr.region = CSL_EDMA3_REGION_1 ;

    // Set IER for channel 15, 14, and load channels.
    reg  = 0x0000C000;
    regh = 0x00000000;
    if (EDMACHAN0 < 32)
            reg |= 1 << EDMACHAN0;
    else
            regh |= 1 << (EDMACHAN0 - 32);
    if (EDMACHAN1 < 32)
            reg |= 1 << EDMACHAN1;
    else
            regh |= 1 << (EDMACHAN1 - 32);
    if (EDMACHAN2 < 32)
            reg |= 1 << EDMACHAN2;
    else
            regh |= 1 << (EDMACHAN2 - 32);
    regionIntr.intr  = reg;
regionIntr.intrh = regh;
CSL_edma3HwControl(hModule,CSL_EDMA3_CMD_INTR_ENABLE,&regionIntr);

/* Loading channel event clear and enable */
CSL_edma3HwChannelControl(hChannelLoad0,CSL_EDMA3_CMD_CHANNEL_CLEAR,NULL);
CSL_edma3HwChannelControl(hChannelLoad0,CSL_EDMA3_CMD_CHANNEL_ENABLE,NULL);
```

```
    CSL_edma3HwChannelControl(hChannelLoad1,CSL_EDMA3_CMD_CHANNEL_CLEAR,NULL);
    CSL_edma3HwChannelControl(hChannelLoad1,CSL_EDMA3_CMD_CHANNEL_ENABLE,NULL);
    CSL_edma3HwChannelControl(hChannelLoad2,CSL_EDMA3_CMD_CHANNEL_CLEAR,NULL);
    CSL_edma3HwChannelControl(hChannelLoad2,CSL_EDMA3_CMD_CHANNEL_ENABLE,NULL);

     /* Transmit event clear and enable */
     CSL_edma3HwChannelControl(hChannelXmt,CSL_EDMA3_CMD_CHANNEL_CLEAR,NULL);
    CSL_edma3HwChannelControl(hChannelXmt,CSL_EDMA3_CMD_CHANNEL_ENABLE,NULL);

     /* Receive event clear and enable */
     CSL_edma3HwChannelControl(hChannelRcv,CSL_EDMA3_CMD_CHANNEL_CLEAR,NULL);
    CSL_edma3HwChannelControl(hChannelRcv,CSL_EDMA3_CMD_CHANNEL_ENABLE,NULL);

    /* Clear CPU interrupt event 72 (EVTCLR2) */
        *((Uint32*)0x1800048) = 0x00000100;

    /* Enable CPU interrupt event 72 (EVTMASK2) */
        *((Uint32*)0x1800088) = 0x00000100;

        /* Re-enable global interrupts */
    HWI_enable();

    /* Manually trigger the loading channels */
    status = CSL_edma3HwChannelControl(hChannelLoad0,CSL_EDMA3_CMD_CHANNEL_SET,NULL);
    status = CSL_edma3HwChannelControl(hChannelLoad1,CSL_EDMA3_CMD_CHANNEL_SET,NULL);
    status = CSL_edma3HwChannelControl(hChannelLoad2,CSL_EDMA3_CMD_CHANNEL_SET,NULL);

    /* Manually trigger the transmit channel */
    status = CSL_edma3HwChannelControl(hChannelXmt,CSL_EDMA3_CMD_CHANNEL_SET,NULL);

    /* Manually trigger the receive channel */
    status = CSL_edma3HwChannelControl(hChannelRcv,CSL_EDMA3_CMD_CHANNEL_SET,NULL);
}


void main()
{
    DSK6455_AIC23_CodecHandle hCodec;

    /* Initialize the board support library, must be called first */
    DSK6455_init();

    /* Initialize LEDs and DIP switches */
    DSK6455_LED_init();
    DSK6455_DIP_init();

    /* Clear buffers */
    memset((void*)gBufferXmtPing,0,sizeof(gBufferXmtPing));
    memset((void*)gBufferXmtPong,0,sizeof(gBufferXmtPong));
    memset((void*)gBufferRcvPing,0,sizeof(gBufferRcvPing));
    memset((void*)gBufferRcvPong,0,sizeof(gBufferRcvPong));

        /* Start the codec */
    hCodec = DSK6455_AIC23_openCodec(0, &config);

        /* Start the EDMA controller */
        edmaInit();
}
```

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

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
RFID	www.ti-rfid.com	Telephony	www.ti.com/telephony
Low Power Wireless	www.ti.com/lpw	Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265