```
/*  ============================================================================
 *   Copyright (c) Texas Instruments Inc 2002, 2003, 2004, 2005, 2008
 *
 *   Use of this software is controlled by the terms and conditions found in the
 *   license agreement under which this software has been supplied.
 *  ============================================================================
 */

/** @file csl_dma_PingPongExample.c
 *
 *  @brief Test DMA ping-pong mode
 *
 *
 * \page    page2  DMA EXAMPLE DOCUMENTATION
 *
 * \section DMA4   DMA EXAMPLE4 - PING-PONG MODE TEST
 *
 * \subsection DMA4x  TEST DESCRIPTION:
 *      This test verifies operation of the CSL DMA module in ping-pong mode.
 * Ping-Pong mode is a special mode of data transfer in which DMA generates
 * a half way interrupt. When ping-pong mode is enabled DMA considers the
 * source data buffer as two parts and generates an interrupt after
 * transferring half of the data. First half of the buffer is Ping data buffer
 * and second half of the buffer is Pong data buffer. These two buffers
 * should be allocated in contiguous memory locations or one ping buffer of size
 * equal to double the data transfer length should be allocated. Buffer being
 * transferred by the DMA currently can be identified by reading
 * 'Last Transfer Type' bit of the channel transfer control register.
 * Ping-Pong mode can be utilized effectively in the interrupt mode of data
 * transfer.
 *
 * During the test DMA ping-pong functionality is verified by transferring
 * the data between four buffers allocated in memory of C5515 DSP. There is
 * one Ping set of data buffers and one Pong set of data buffers. DMA is
 * configured for ping-pong mode and data transfer is started. Execution will
 * wait for the ping and pong data transfer interrupts to occur. Successful
 * completion of data transfer will copy Ping source buffer data into Ping
 * destination buffer and Pong source buffer data into Pong destination buffer.
 * Status of the Ping and Pong set transfer will be displayed in the CCS stdout
 * window. Test is repeated for all the 16 DMA channels.
 *
 * NOTE: DMA PING-PONG BUFFER MODE IS SUPPORTED ONLY ON CHIP 5515. THIS TEST HAS
 * BEEN DEVELOPED TO WORK WITH CHIP C5515. MAKE SURE THAT CHIP VERSION MACRO
 * 'CHIP_5515' IS DEFINED IN THE FILE c55xx_csl\inc\csl_general.h.
 *
 * \subsection DMA4y TEST PROCEDURE:
 *  @li Open the CCS and connect the target (C5515 EVM)
 *  @li Open the project "CSL_DMA_PingPongExample.pjt" and build it
 *  @li Load the program on to the target
 *  @li Set the PLL frequency to 12.288MHz
 *  @li Run the program and observe the test result
 *  @li Repeat the test at PLL frequencies 40, 60, 75 and 100MHz
 *  @li Repeat the test in Release mode
 *
 * \subsection DMA4z TEST RESULT:
 *  @li All the CSL APIs should return success
 *  @li Ping source and destination data should match on all the 16 DMA channels
 *  @li Pong source and destination data should match on all the 16 DMA channels
 *
 *
 */

/* ============================================================================
 * Revision History
 * ================
 * 28-Dec-2009 Created
 * ============================================================================
 */
```

```
#include "csl_dma.h"
#include "csl_intc.h"
#include "usbstk5515.h"
#include <stdio.h>

#define CSL_DMA_BUFFER_SIZE 512

/* Reference the start of the interrupt vector table */
extern void VECSTART(void);
/* prototype declaration for ISR function */

/**
 *  \brief  DMA Interrupt Service routine
 *
 *  \param  none
 *
 *  \return none
 */
interrupt void dma_isr(void);

//#pragma DATA_SECTION (dmaPingDstBuf,".daram_buf")
#pragma DATA_ALIGN (dmaPingDstBuf, 4)
Uint16 dmaPingDstBuf[CSL_DMA_BUFFER_SIZE];

//#pragma DATA_ALIGN (dmaPongDstBuf, 4)
//Uint16 dmaPongDstBuf[CSL_DMA_BUFFER_SIZE];

/* Declaration of the buffer */
//#pragma DATA_ALIGN (dmaPingSrcBuf, 4)
//Uint16 dmaPingSrcBuf[CSL_DMA_BUFFER_SIZE];

//#pragma DATA_ALIGN (dmaPongSrcBuf, 4)
//Uint16 dmaPongSrcBuf[CSL_DMA_BUFFER_SIZE];

static int count ;
static int isrEntryCount = 0;

CSL_DMA_Handle      dmaHandleI2s;
CSL_DMA_Config      dmaConfig;
CSL_DMA_Config      getdmaConfig;

CSL_Status          status;

/**
 *  \brief  Tests DMA Ping-Pong Mode transfers
 *
 *  \param  none
 *
 *  \return none
 */
   /////INSTRUMENTATION FOR BATCH TESTING -- Part 1 --
   /////  Define PaSs_StAtE variable for catching errors as program executes.
   /////  Define PaSs flag for holding final pass/fail result at program completion.
        volatile Int16 PaSs_StAtE2 = 0x0001; // Init to 1. Reset to 0 at any monitored
execution error.
        volatile Int16 PaSs2 = 0x0000; // Init to 0.  Updated later with PaSs_StAtE
when and if
   /////                                  program flow reaches expected exit point(s).
   /////
void Config_DMA_I2S(void)
{
#if ((defined(C5515_EZDSP)) || (defined(CHIP_5514)))

    CSL_DMA_ChannelObj  dmaObj;
    Uint16              chanNumber;
    Uint16              index;
```

```
    printf("\nDMA PING-PONG MODE TEST!\n");

    dmaConfig.pingPongMode = CSL_DMA_PING_PONG_ENABLE;
    dmaConfig.autoMode     = CSL_DMA_AUTORELOAD_ENABLE;
    dmaConfig.burstLen     = CSL_DMA_TXBURST_1WORD;
    dmaConfig.trigger      = CSL_DMA_EVENT_TRIGGER;
    dmaConfig.dmaEvt       = CSL_DMA_EVT_I2S0_RX;
    dmaConfig.dmaInt       = CSL_DMA_INTERRUPT_ENABLE;
    dmaConfig.chanDir      = CSL_DMA_READ;
    dmaConfig.trfType      = CSL_DMA_TRANSFER_IO_MEMORY;
    dmaConfig.dataLen      = CSL_DMA_BUFFER_SIZE * 2;
    dmaConfig.srcAddr      = (Uint32)0x2828; // I2SRXLT0
    dmaConfig.destAddr     = (Uint32)dmaPingDstBuf; // DARAM1

    IRQ_globalDisable();

    IRQ_clearAll();

    IRQ_disableAll();

    IRQ_setVecs((Uint32)&VECSTART);
    IRQ_clear(DMA_EVENT);

    IRQ_plug (DMA_EVENT, &dma_isr);

    IRQ_enable(DMA_EVENT);
    IRQ_globalEnable();

    status = DMA_init();
    if (status != CSL_SOK)
    {
        printf("DMA_init() Failed \n");
   /////INSTRUMENTATION FOR BATCH TESTING -- Part 2 --
   /////  Reseting PaSs_StAtE to 0 if error detected here.
        PaSs_StAtE2 = 0x0000; // Was intialized to 1 at declaration.
   /////
    }
    //for( chanNumber = 0; chanNumber < CSL_DMA_CHAN_MAX; chanNumber++)
    //{
        count = 0;
        printf("\nTest for DMA Channel Number: %d\n", chanNumber);

        for(index = 0; index < CSL_DMA_BUFFER_SIZE*2; index++)
        {
            //dmaPingSrcBuf[index]  = index;
            //dmaPongSrcBuf[index]  = 2*index;

            dmaPingDstBuf[index] = 0x0000;
            //dmaPongDstBuf[index] = 0x0000;
        }

        dmaHandleI2s = DMA_open(CSL_DMA_CHAN1,&dmaObj, &status);
        if (dmaHandleI2s == NULL)
        {
            printf("DMA_open() Failed \n");
            //break;
        }

        status = DMA_config(dmaHandleI2s, &dmaConfig);
        if (status != CSL_SOK)
        {
            printf("DMA_config() Failed \n");
            //break;
        }

        status = DMA_start(dmaHandleI2s);
        if (status != CSL_SOK)
        {
```

```
            printf("DMA_start() Failed \n");
            //break;
        }

        /*while(count != 2);

        status = DMA_close(dmaHandleI2s);
        if (status != CSL_SOK)
        {
            printf("DMA_close() Failed \n");
            //break;
        }

        status = DMA_reset(dmaHandleI2s);
        if (status != CSL_SOK)
        {
            printf("DMA_reset() Failed \n");
            //break;
        }

        for(index = 0; index < CSL_DMA_BUFFER_SIZE; index++)
        {
            if(dmaPingSrcBuf[index] != dmaPingDstBuf[index])
            {
                printf("Ping Buffer Miss Matched at Position %d\n", index);
                break;
            }

            if(dmaPongSrcBuf[index] != dmaPongDstBuf[index])
            {
                printf("Pong Buffer Miss Matched at Position %d\n", index);
                break;
            }
        }

        if(index == CSL_DMA_BUFFER_SIZE)
        {
            printf("Test Successful\n");
        }*/
    //}

    //IRQ_clearAll();
    //IRQ_disableAll();
    //IRQ_globalDisable();

    if(isrEntryCount == 2)
    {
        printf("\n\nDMA PING-PONG MODE TEST PASSED!!\n");
    }
    else
    {
        printf("\n\nDMA PING-PONG MODE TEST FAILED!!\n");
   /////INSTRUMENTATION FOR BATCH TESTING -- Part 2 --
   /////  Reseting PaSs_StAtE to 0 if error detected here.
        PaSs_StAtE2 = 0x0000; // Was intialized to 1 at declaration.
   /////
    }

#else

    printf("\n\nINVALID TEST FOR THE CHIP VERSION!!\n");
   /////INSTRUMENTATION FOR BATCH TESTING -- Part 2 --
   /////  Reseting PaSs_StAtE to 0 if error detected here.
        PaSs_StAtE2 = 0x0000; // Was intialized to 1 at declaration.
   /////

#endif
   /////INSTRUMENTATION FOR BATCH TESTING -- Part 3 --
```

```
   /////  At program exit, copy "PaSs_StAtE" into "PaSs".
        PaSs2 = PaSs_StAtE2; //If flow gets here, override PaSs' initial 0 with
   /////                     // pass/fail value determined during program execution.
   /////  Note:  Program should next exit to C$$EXIT and halt, where DSS, under
   /////   control of a host PC script, will read and record the PaSs' value.
   /////
}

/**
 *  \brief  DMA Interrupt Service routine
 *
 *  \param  none
 *
 *  \return none
 */
interrupt void dma_isr(void)
{
    int ifrValue;

    ifrValue = CSL_SYSCTRL_REGS->DMAIFR;
    CSL_SYSCTRL_REGS->DMAIFR |= ifrValue;

#if ((defined(C5515_EZDSP)) || (defined(CHIP_5514)))

    /*if ((DMA_getLastTransferType (dmaHandleI2s, &status)) == 1)
    {
        printf("Pong Set Transfer Completed\n");
    }
    else
    {
        printf("Ping Set Transfer Completed\n");
    }*/

#endif

    ++count;
    ++isrEntryCount;
}
```