```
//////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////
/// LABMAIN.C --- DMA TEMPLATE
//////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////

#include "shared.h"
#include <math.h> /* Note: We get a warning if math.h is moved up */



#define SZ_TABLE 512

void initEdma( int *table);
void create_table( int *table);

void labmain(void)
{
    static int table[SZ_TABLE]; // DMA Controller will read from this
                           // This variable must be static or global

    create_table(table);        // Initialize table of samples

    initEdma(table);        // Initialize the EDMA controller

    while(1);           /* infinite loop, DMA Controller does rest  */
}


/*******************************************************/
/* Create a table where upper 16-bits are samples of   */
/* a sine wave with frequency f_left, and the lower 16 */
/* bits are samples of a sine wave with frequency      */
/* f_right.                                            */
/*******************************************************/

void create_table(int *table){
    /* Initialize the table with samples            */
    /* Each entry is a 32-bit packet integer, with  */
    /* the high and low 16 bits representing a      */
        /* signed sample value. See pages 50-51 in Tretter */
}

/*
 *  initEdma() - Initialize the DMA controller.  Use linked transfers to
 *               automatically restart at beginning of sine table.
 */

void initEdma(int *table)
{

    EDMA_Config gEdmaConfigXmt;
    EDMA_Handle hEdmaXmt;            // EDMA channel handles
    EDMA_Handle hEdmaReloadXmt;

    /* Transmit side EDMA configuration */
    gEdmaConfigXmt.opt =
        EDMA_FMKS(OPT, PRI, HIGH)          |  // Priority
        EDMA_FMKS(OPT, ESIZE, 32BIT)       |  // Element size
        EDMA_FMKS(OPT, 2DS, NO)            |  // 1 dimensional source
        EDMA_FMKS(OPT, SUM, INC)           |  // Src update mode
        EDMA_FMKS(OPT, 2DD, NO)            |  // 1 dimensional dest
        EDMA_FMKS(OPT, DUM, NONE)          |  // Dest update mode
        EDMA_FMKS(OPT, TCINT, NO)          |  // Cause EDMA interrupt?
        EDMA_FMKS(OPT, TCC, OF(0))         |  // Transfer complete code
        EDMA_FMKS(OPT, LINK, YES)          |  // Enable link parameters?
        EDMA_FMKS(OPT, FS, NO);               // Use frame sync?

    gEdmaConfigXmt.src =  (Uint32)table;      //  Src address
```

```
    gEdmaConfigXmt.cnt =
        EDMA_FMK (CNT, FRMCNT, NULL)       |  // Frame count
        EDMA_FMK (CNT, ELECNT, SZ_TABLE);     // Element count

    gEdmaConfigXmt.dst =
        EDMA_FMKS(DST, DST, OF(0));           // Dest address

    gEdmaConfigXmt.idx =
        EDMA_FMKS(IDX, FRMIDX, DEFAULT)   |   // Frame index value
        EDMA_FMKS(IDX, ELEIDX, DEFAULT),      // Element index value

    gEdmaConfigXmt.rld =
        EDMA_FMK (RLD, ELERLD, NULL)       |  // Reload element
        EDMA_FMK (RLD, LINK, NULL);           // Reload link

    // get hEdmaXmt handle and reset channel for McBSP1 writes
    hEdmaXmt = EDMA_open(EDMA_CHA_XEVT1, EDMA_OPEN_RESET);

    // get handle for reload table
    hEdmaReloadXmt = EDMA_allocTable(-1);

    // Get the address of DXR for McBSP1
    gEdmaConfigXmt.dst = MCBSP_getXmtAddr(DSK6713_AIC23_DATAHANDLE);

    // then configure the Xmt table
    EDMA_config(hEdmaXmt, &gEdmaConfigXmt);

    // Configure the Xmt reload table
    EDMA_config(hEdmaReloadXmt, &gEdmaConfigXmt);

    // link back to table start
    EDMA_link(hEdmaXmt,hEdmaReloadXmt);
    EDMA_link(hEdmaReloadXmt, hEdmaReloadXmt);

    // enable EDMA channel
    EDMA_enableChannel(hEdmaXmt);

    /* Do a dummy write to generate the first McBSP transmit event */
    WriteSample(0,0);
}
```