```
/*
Program block_dma_audio_asm demonstrates a simple DMA-based audio in-out capability of the ADSP-BF706
EZ-KIT Mini evaluation system using an interrupt-driven algorithm. The program is written in assembly
code, so is optimized for high speed operation. It is also entirely self-contained, using only the
header defBF706.h for the addresses of the BF706 registers. Note that it is a very basic skeleton,
and has no error checking, so should only be used as a base to develop more complex code.

Acronyms as follows:

BHRM: ADSP-BF70x Blackfin+ Processor Hardware Reference Manual, version 1.0
BPRM: ADSP-BF70x Blackfin+ Processor Programming Reference Manual, version 1.0

Author: Patrick Gaydecki
Date : 07.05.2021
*/

.section program;
.align 4;
.global _main;
#include <defBF706.h>
_main:
  call codec_configure;
  call sport_configure;
  call DMA_init;
  P0=0x1;
  [REG_PORTC_DIR_SET]=P0;                   // Set GPIO. Debug only
  P0=0x81F; [IMASK]=P0;                     // Enable interrupt 11 via the IMASK register
  P0=(get_audio); [EVT11]=P0;               // Get the address of the ISR
  P0=0x5; [REG_SEC0_SCTL31]=P0;             // Enable source control register 31 for DMA interrupt
  [REG_SEC0_GCTL]=P0;                       // Enable global interrupts
  [REG_SEC0_CCTL0]=P0;                      // Enable core interrupts
  idler:                                    // Now do nothing
//  call pulser;                            // Diagnostic only. If enabled, shows mostly idle
  jump idler;
._main.end:

/*This is the ISR which handles DMA data in/out. The DMA buffer is here 4-words (2 left, 2 right
channel), but is easily modified Resetting of the interrupts regarding CEC_SID and REG_SEC0_END is
described on p4-44, BPRM. Note: ALL hardware system interrupts vector to the address held in register
EVT11 (p4-35, BPRM), so this program can only handle one interrupt. More ISR branch code would be
required in the ISR to handle interrupts from multiple sources.*/

get_audio:
  P5=[CEC_SID];                             // Store interrupt source
  [CEC_SID]=R0;                             // Acknowledge interrupt by writing any value
  P0=0x1;
  [REG_DMA1_STAT]=P0;
  R0=[0x11900000]; [0x11800000]=R0;         // Rx the data and Tx the data
  R0=[0x11900004]; [0x1180000c]=R0;
  R0=[0x11900008]; [0x11800008]=R0;
  R0=[0x1190000c]; [0x11800004]=R0;
  call pulser;                              // 2 words per channel, pulsing = 24 kHz on pin PC0
  [REG_SEC0_END]=P5;                        // Acknowledge interrupt has been serviced
  rti;
get_audio.end:

// Subroutine DMA_init initialises the SPORT0 DMA0 and DMA1 in autobuffer mode, p19-39 and p19-49,
// BHRM.

DMA_init:
  P0=0x11800000; [REG_DMA0_ADDRSTART]=P0;   // points to start of SPORT0_A buffer
  P0=0x4;        [REG_DMA0_XCNT]=P0;        // No. of words to transmit. This could be any length
  P0=0x4;        [REG_DMA0_XMOD]=P0;        // Word length, i.e. step increment to next word
  P0=0x11900000; [REG_DMA1_ADDRSTART]=P0;   // Points to start of SPORT0_B buffer
  P0=0x4;        [REG_DMA1_XCNT]=P0;        // No. of words to receive. This could be any length
  P0=0x4;        [REG_DMA1_XMOD]=P0;        // Word length, i.e. step increment to next word
  P0=0x00001221; [REG_DMA0_CFG]=P0;         // SPORT0 TX, FLOW=autobuffer, MSIZE=PSIZE=4 bytes
  P0=0x00101223; [REG_DMA1_CFG]=P0;         // SPORT0 RX, DMA interrupt when x count expires
  rts;
DMA_init.end:

// Subroutine codec_configure initialises the ADAU1761 codec. Refer to the control register
// descriptions, page 51 onwards of the ADAU1761 data sheet.

codec_configure:
  [--SP] = RETS;                            // Push stack (only for nested calls)
  R1=0x01(X); R0=0x4000(X); call TWI_write; // Enable master clock, disable PLL
  R1=0x7f(X); R0=0x40f9(X); call TWI_write; // Enable all clocks
  R1=0x03(X); R0=0x40fa(X); call TWI_write; // Enable all clocks
  R1=0x01(X); R0=0x4015(X); call TWI_write; // Set serial port master mode
```

```
  R1=0x13(X); R0=0x4019(X); call TWI_write; // Set ADC to on, both channels
  R1=0x21(X); R0=0x401c(X); call TWI_write; // Enable left channel mixer
  R1=0x41(X); R0=0x401e(X); call TWI_write; // Enable right channel mixer
  R1=0x03(X); R0=0x4029(X); call TWI_write; // Turn on power, both channels
  R1=0x03(X); R0=0x402a(X); call TWI_write; // Set both DACs on
  R1=0x01(X); R0=0x40f2(X); call TWI_write; // DAC gets L, R input from serial port
  R1=0x01(X); R0=0x40f3(X); call TWI_write; // ADC sends L, R input to serial port
  R1=0x0b(X); R0=0x400a(X); call TWI_write; // Set left line-in gain to 0 dB
  R1=0x0b(X); R0=0x400c(X); call TWI_write; // Set right line-in gain to 0 dB
  R1=0xe7(X); R0=0x4023(X); call TWI_write; // Set left headphone volume to 0 dB
  R1=0xe7(X); R0=0x4024(X); call TWI_write; // Set right headphone volume to 0 dB
  R1=0x00(X); R0=0x4017(X); call TWI_write; // Set codec default sample rate, 48 kHz
  nop;
  RETS = [SP++];                            // Pop stack (only for nested calls)
  rts;
codec_configure.end:

// Subroutine sport_configure initialises the SPORT0. Refer to pages 20-31, 20-42, 31-55,
// 31-63, 31-72 and 31-73 of the BHRM.

sport_configure:
  R0=0x003F0(X); [REG_PORTC_FER]=R0;        // Set up Port C in peripheral mode
  R0=0x003F0(X); [REG_PORTC_FER_SET]=R0;    // Set up Port C in peripheral mode
  R0=0x02001973; [REG_SPORT0_CTL_A]=R0;     // Set up SPORT0 (A) as TX to codec, 24 bits
  R0=0x00400001; [REG_SPORT0_DIV_A]=R0;     // 64 bits per frame, clock divisor of 1
  R0=0x00001973; [REG_SPORT0_CTL_B]=R0;     // Set up SPORT0 (B) as RX from codec, 24 bits
  R0=0x00400001; [REG_SPORT0_DIV_B]=R0;     // 64 bits per frame, clock divisor of 1
  rts;
sport_configure.end:

// Subroutine TWI_write is a simple driver for the TWI. Refer to page 26-15 onwards, BHRM.

TWI_write:
  R3=R0 <<0x8; R0=R0 >>>0x8; R2=R3|R0;      // Reverse low order and high order bytes
  R0=0x3232(X); [REG_TWI0_CLKDIV]=R0;       // Set duty cycle
  R0=0x008c(X); [REG_TWI0_CTL]=R0;          // Set pre-scale and enable TWI
  R0=0x0038(X); [REG_TWI0_MSTRADDR]=R0;     // Address of codec
  [REG_TWI0_TXDATA16]=R2;                   // Address of register to set, LSB then MSB
  R0=0x00c1(X); [REG_TWI0_MSTRCTL]=R0;      // Command to send three bytes and enable TX
  [--SP] = RETS; call delay; RETS = [SP++]; // Delay
  [REG_TWI0_TXDATA8]=R1;                    // Data to write
  [--SP] = RETS; call delay; RETS = [SP++]; // Delay
  R0=0x050; [REG_TWI0_ISTAT]=R0;            // Clear TXERV interrupt
  [--SP] = RETS; call delay; RETS = [SP++]; // Delay
  R0=0x010; [REG_TWI0_ISTAT]=R0;            // Clear MCOMP interrupt
  rts;
TWI_write.end:

// Subroutine delay introduces a delay to allow TWI communication.

delay:
  P0=0x8000;
  loop lc0=P0;
    nop; nop; nop;
  loop_end;
  rts;
delay.end:

// Subroutine pulser pulses the GPIO line, pin 4 on connector P2 of the EZ MINI. Debug only.

pulser:
  P1=0x1;
  loop lc0 = 40;
    [REG_PORTC_DATA]=P1;
  loop_end;
  P1=0;
  loop lc0 = 40;
    [REG_PORTC_DATA]=P1;
  loop_end;
  rts;
pulser.end:
```