```
Program interrupt_inout demonstrates a simple audio in-out capability of the ADSP-BF706
EZ-KIT Mini evaluation system using an interrupt-driven, sample-by-sample 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.

Author: Patrick Gaydecki
Date  : 16.12.2019
*/

.section program;
.align 4;
.global _main;
#include <defBF706.h>

_main:
call codec_configure;
call sport_configure;
nop;
p0=0x1;
// Enable global and core interrupts
[REG_SEC0_GCTL]=p0;
[REG_SEC0_CCTL0]=p0;
// Enable interrupt 11 via the IMASK register
p0=0x81F;
[IMASK]=p0;
// Get the address of the interrupt service routine (ISR)
p0=(get_audio);
[EVT11]=p0;
//  Now enable the source control register 31. This responds to an
// interrupt from the DMA.
p0=0x5;
[REG_SEC0_SCTL31]=p0;
// Now do nothing
idler:
nop;
jump idler;

// Main ISR
get_audio:
R0=[REG_SPORT0_RXPRI_B]; [REG_SPORT0_TXPRI_A]=R0; // Write left out
R0=[REG_SPORT0_RXPRI_B]; [REG_SPORT0_TXPRI_A]=R0; // Write right out
// The next line restores bit 11 in the ILAT register, which is automatically
// cleared when the ISR is activated. This is required.
raise 0xb;
rti;
._main.end:

// Function 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:

// Function sport_configure initialises the SPORT0. Refer to pages 20-31, 20-42, 31-55,
// 31-63, 31-72 and 31-73 of the ADSP-BF70x Blackfin+ Processor Hardware Reference
manual.
sport_configure:
R0=0x3F0(X); [REG_PORTC_FER]=R0;          // Set up Port C in peripheral mode
R0=0x3F0(X); [REG_PORTC_FER_SET]=R0;      // Set up Port C in peripheral mode
R0=0x2001973; [REG_SPORT0_CTL_A]=R0;      // Set up SPORT0 (A) as TX to codec, 24 bits
R0=0x0400001; [REG_SPORT0_DIV_A]=R0;      // 64 bits per frame, clock divisor of 1
R0=0x1973(X); [REG_SPORT0_CTL_B]=R0;      // Set up SPORT0 (B) as RX frm codec, 24 bits
R0=0x0400001; [REG_SPORT0_DIV_B]=R0;      // 64 bits per frame, clock divisor of 1
rts;
sport_configure.end:

// Function TWI_write is a simple driver for the TWI. Refer to page 26-15 onwards
// of the ADSP-BF70x Blackfin+ Processor Hardware Reference manual.
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:

// Function delay introduces a delay to allow TWI communication
delay:
p0=0x8000;
loop lc0=p0;
nop; nop; nop;
loop_end;
rts;
delay.end:
```

```
/*
Program C_inout_int is a simple audio in-out program for the ADSP-BF706 EZ-KIT Mini
evaluation system. It uses interrupts to acquire and transmit the data, sample-by-sample.
It uses a function to install the handler.

Author: Patrick Gaydecki
Date  : 16.12.2019
*/

#include <sys/platform.h>
#include "adi_initialize.h"
#include <cdefBF706.h>
#include <services/int/adi_int.h>

void TWI_write(uint16_t, uint8_t);
void codec_configure(void);
void sport_configure(void);

// Function sport_configure initialises the SPORT0. Refer to pages 20-31, 20-42, 31-55,
// 31-63, 31-72 and 31-73 of the ADSP-BF70x Blackfin+ Processor Hardware Reference
manual.
void sport_configure()
{
  *pREG_PORTC_FER=0x0003F0;             // Set up Port C in peripheral mode
  *pREG_PORTC_FER_SET=0x0003F0;         // Set up Port C in peripheral mode
  *pREG_SPORT0_CTL_A=0x2001973;         // Set up SPORT0 (A) as TX to codec, 24 bits
  *pREG_SPORT0_DIV_A=0x400001;          // 64 bits per frame, clock divisor of 1
  *pREG_SPORT0_CTL_B=0x0001973;         // Set up SPORT0 (B) as RX from codec, 24 bits
  *pREG_SPORT0_DIV_B=0x400001;          // 64 bits per frame, clock divisor of 1
}

// Function TWI_write is a simple driver for the TWI. Refer to page 24-15 onwards
// of the ADSP-BF70x Blackfin+ Processor Hardware Reference manual.
void TWI_write(uint16_t reg_add, uint8_t reg_data)
{
  int n;
  reg_add=(reg_add<<8)|(reg_add>>8);    // Reverse low order and high order bytes
  *pREG_TWI0_CLKDIV=0x3232;             // Set duty cycle
  *pREG_TWI0_CTL=0x8c;                  // Set prescale and enable TWI
  *pREG_TWI0_MSTRADDR=0x38;             // Address of codec
  *pREG_TWI0_TXDATA16=reg_add;          // Address of register to set, LSB then MSB
  *pREG_TWI0_MSTRCTL=0xc1;              // Command to send three bytes and enable tx
  for(n=0;n<8000;n++){}                 // Delay since codec must respond
  *pREG_TWI0_TXDATA8=reg_data;          // Data to write
  for(n=0;n<10000;n++){}                // Delay
  *pREG_TWI0_ISTAT=0x050;               // Clear TXERV interrupt
  for(n=0;n<10000;n++){}                // Delay
  *pREG_TWI0_ISTAT=0x010;               // Clear MCOMP interrupt
}

// Function codec_configure initialises the ADAU1761 codec. Refer to the control register
// descriptions, page 51 onwards of the ADAU1761 data sheet.
void codec_configure()
{
  TWI_write(0x4000, 0x01);              // Enable master clock, disable PLL
  TWI_write(0x40F9, 0x7f);              // Enable all clocks
  TWI_write(0x40Fa, 0x03);              // Enable all clocks
  TWI_write(0x4015, 0x01);              // Set serial port master mode
  TWI_write(0x4019, 0x13);              // Set ADC to on, both channels
  TWI_write(0x401c, 0x21);              // Enable left channel mixer
  TWI_write(0x401e, 0x41);              // Enable right channel mixer
  TWI_write(0x4029, 0x03);              // Turn on power, both channels
  TWI_write(0x402A, 0x03);              // Set both DACs on
  TWI_write(0x40f2, 0x01);              // DAC gets L, R input from serial port
  TWI_write(0x40f3, 0x01);              // ADC sends L, R input to serial port
  TWI_write(0x400a, 0x0b);              // Set left line-in gain to 0 dB
  TWI_write(0x400c, 0x0b);              // Set right line-in gain to 0 dB
  TWI_write(0x4023, 0xe7);              // Set left headphone volume to 0 dB
  TWI_write(0x4024, 0xe7);              // Set right headphone volume to 0 dB
  TWI_write(0x4017, 0x00);              // Set codec default sample rate, 48 kHz
}
```

```
// The ISR
void SPORT_ISR(uint32_t iid, void* handlerArg)
{
    *pREG_SPORT0_TXPRI_A=*pREG_SPORT0_RXPRI_B;       // Read right channel in, send out
    *pREG_SPORT0_TXPRI_A=*pREG_SPORT0_RXPRI_B;       // Read right channel in, send out
}

int main(void)
{
  bool my_audio=true;
//Initialize managed drivers and/or services
  codec_configure();
  sport_configure();
// Global and core interrupt enable
  *pREG_SEC0_GCTL=1;
  *pREG_SEC0_CCTL0=1;
// Install interrupt handler (from adi_int.h).
  adi_int_InstallHandler(INTR_SPORT0_B_DMA, SPORT_ISR, NULL, true);
// Now loop forever, waiting for interrupt.
  while(my_audio);
  return 0;
}
```
