```
/*
Program FIR4 is a simple C program that uses embedded assembly to perform 32-bit filtering.
Because it uses in line assembly code for the convolution, it is highly efficient, allowing
a filter length of 8060 taps at a sample rate of 48 kHz. This represents an efficiency of
97% theoretical maximum. It uses a function called memwrite to pass data from the C code
to the assembly.

Author: Patrick Gaydecki
Date  : 01.10.2019
*/

#include <stdio.h>
#include <cdefBF706.h>
#include <defBF706.h>

#define codec_in  ((volatile long fract *)0x2004D0C4)
#define codec_out ((volatile long fract *)0x2004D040)

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

// Function memwrite writes a 32-bit value to a memory location
// Inputs: address: the 32-bit address; memvalue: 32-bit value to write
void memwrite(unsigned address, unsigned memvalue)
{
   unsigned *towrite;
   towrite = (unsigned*)address;
   *towrite=memvalue;
}

// Function sport_configure initialises the SPORT0. Refer to pages 26-59, 26-67,
// 26-75 and 26-76 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
}

#pragma optimize_for_speed

int main(void)
{
  unsigned filter_length=8060;
// Set the clock to maximum
  *pREG_CGU0_CTL=0x2000;
  *pREG_CGU0_DIV=0X42042442;
  codec_configure();                    // Configure the codec
  sport_configure();                    // Configure SPORT0
  memwrite(0x08000000, filter_length);  // Write filter length to L2 memory
  memwrite(0x08000004, filter_length*4); // Write X4 length to memory
  asm                                   // In line assembly starts here
  (
       "P0=[0x08000000];"               // Load filter length into P0
       "R0.h=0.001r;"                   // Next 4 lines generate filter coefficients
       "I1=0x11900000;"
       "loop LC0=P0;"
       "[I1++]=R0;"
       "loop_end;"
       "P0=[0x08000004];"           // Circular buffer in bytes (hence X4)
       "I0=0x11800000;B0=I0;L0=P0;" // Circular buffer for input data, Block A
       "I1=0x11900000;B1=I1;L1=P0;" // Circular buffer for filter coefficients, Block B
       "P0=[0x08000000];P0+=-1;"    // Decrement by 1 for circular addressing
       "get_audio:"
       "wait_left:"
       "R0=[0x2004D080];CC=BITTST(R0, 31);if !CC jump wait_left;"   // Wait left flag in
       "R0=[0x2004D0C4];"                                   // Get value
       "A0 = 0 || [I0++]= R0 || R1 = [I1++];"               // Clear A0 and get data/taps
       "A1=0;"                                              // Also clear A1
       "LOOP LC0=P0;"
       "A1:0+= R0 * R1 || R0 = [I0++] || R1 = [I1++];"      // Filter left
       "LOOP_END;"
       "A1:0+= R0*R1 || I0-=4;"
       "R1=A1:0;[0x2004D040]=R1;"                           // Send left data to codec
       "wait_right:"
       "R0=[0x2004D080];CC=BITTST(R0, 30);if !CC jump wait_right;"  // Wait right flag in
       "R0=[0x2004D0C4];[0x2004D040]=R0;"                           // Send right to codec
      "jump get_audio;"
  );
  return 0;
}
```