ADSP-BF706 Assembly Programming Examples

```
/*
 Program ASM 1 tests various simple assembly language statements:

 1. 16 and 32-bit arithmetic
 2. Parallel instructions
 3. Constants
 4. Numbering formats
 5. Store and retrieve from memory
 6. Loops and conditionals
 7. Bit set/clear/toggle
 8. Arithmetic shifts

 Author: Patrick Gaydecki
 Date  : 24.09.2016
 */
.section program;
.align 4;
.global _main;
.extern _adi_initComponents;
#define bb1 0.5r // Define a constant

_main:
//   call _adi_initComponents;
// 16-bit add and multiply.
     r0=0;r1=0;r2=0;r3=0;
    r0.l=8192;
    r1.l=8192;
    r2.l=r0.l+r1.l;
    r3=r0*r1;
    r3=r0.l*r1.l;
    a0=r0.l*r1.l;
// 32-bit add and multiply.
     r0=0;r1=0;r2=0;r3=0;
    r0.h=8192;
    r1.h=8192;
    r2=r0+r1;
    r3=r0*r1;
    a0=r0.h*r1.h;
// 16-bit add, multiply, increment and parallel move.
     r0=0;r1=0;r2=0;r3=0;
     p0=9;
    r0.l=8192;
    r1.l=b#101;         // The #b indicates a bianry value.
    r1.l=0.125r;        // The r suffix indicates fractional value.
    r1.l=bb1;
    r1=0;
    r1=bb1;             // This is the same as r1.l=bb1;
    r1.h=bb1;
    r2.l=r0.l+r1.l;
    A0=r0.l*r1.l;
    A0+=r0.l*r1.l||r2 = [i0++];  // Parallel multiply and move.
// Now test multiplier instruction options
     r0=0;r1=0;r2=0;r3=0;
     r0=0x100;
     r0=0.2r;
     r1=0.3r;
     r2=r0.l*r1.l;       // This gives r2=0.06
     r2=r0.l*r1.l(FU);   // This gives r2=0.03
     r0=100;r1=250;
     r2=r0+r1;           // This gives r2=15e
// Memory move operations.
     r0=0;r1=0;r2=0;r3=0;
    p0=0x11800000;
    r0=0.34719r;
```

```
    [p0]=r0;            // Load to memory
    r1=[p0];            // Retrieve data from memory
    [p0++]=r0;          // This increments by 4, since it is a 32-bit word
// Now test loop structures.
    r0=0;
    loop lc0=10;
    r0+=1;
    loop_end;
// Now test condition code bit cc.
    a0=0;a1=0;
    r0=0.2r;
    r1=0.3r;
    cc=r0<r1;           // If r0<r1, set cc to true, (i.e. 1)
    if cc jump lq1;     // Jump to lq1 if cc=1
     nop;                // Jump instruction
lq1:
    r0=0.67r;
    cc=r1<r0;           // If r0<r1, set cc to true, (i.e. 1)
    if cc jump lq2;     // Jump to lq1 if cc=1
     nop;               // Jump instruction
lq2:
// Now test a bit set/clear/toggle operations.
    r0=0;
    bitset (r0,7);
    bitclr(r0,7);
    bittgl (r0, 24);
    bittgl (r0, 24);
    nop;
// Now test arithmetic shift operations.
    r0=0;r1=0;r2=0;
    r0.l=b#110000;
    r0.l=r0.l>>>3;      // Arithmetic right shift
    r0.l=r0.l<<2;       // Arithmetic left shift
    r2.l=b#10;
// Now place in r1.l contents of r0.l A-shifted by contents of r2.l.
    r1.l=ashift r0.l by r2.l;
    rts;
._main.end:
```

```
/*
Program ASM 2 flashes LED0, which is connected to pin 3 of the GPIO port C
(PC3).
It takes #define statements from the header file "defBF706.h". Note it only uses
two, i.e. the port direction register (REG_PORTC_DIR_SET) and the port data
register (REG_PORTC_DATA). First, it enables output on pin3 of port C.
Next it enters a loop to pulse the pin. Note: the pit must be low for the LED
to turn (since the LED's cathode is connected to the pin). There is an error in
the
EVM manual, which states the pin must be high.

Author: Patrick Gaydecki
Date  : 01.10.2016
 */

.section program;
.align 4;
.global _main;
.extern _adi_initComponents;

#define REG_PORTC_DIR_SET               0x2004011C         /* PORTC Port x GPIO
Direction Set Register */
#define REG_PORTC_DATA                  0x2004010C         /* PORTC Port x GPIO
Data Register */

_main:
    r0=0;
    r1=b#1000;
// Set the direction and data registers.
    p0=REG_PORTC_DIR_SET;
    p1=REG_PORTC_DATA;
// Set bit 3, port c as output.
    [p0]=r1;
// Endless loop.
flash:
    p3=1000000;
    loop lc0=p3; //delay
    nop;
// Turn on LED0.
    [p1]=r0;
    loop_end;
    p3=1000000;
    loop lc0=p3; //delay
    nop;
// Turn off LED0.
    [p1]=r1;
    loop_end;
    jump flash;
    rts;
._main.end:
```

```
/*
Program ASM 3 pulses pin 0 of the GPIO port C (PC3).
It takes #define statements from the header file "defBF706.h". Note it only uses
two, i.e. the port direction register (REG_PORTC_DIR_SET) and the port data
register (REG_PORTC_DATA). First, it enables output on pin 0 of port C.
Next it enters a loop to pulse the pin.

Author: Patrick Gaydecki
Date  : 03.10.2016
 */
.section program;
.align 4;
.extern _adi_initComponents;

.global _main;

#define REG_PORTC_DIR_SET               0x2004011C         /* PORTC Port x GPIO
Direction Set Register */
#define REG_PORTC_DATA                  0x2004010C         /* PORTC Port x GPIO
Data Register */

_main:
    r0=0;
    r1=b#1;
// Set the direction and data registers.
    p0=REG_PORTC_DIR_SET;
    p1=REG_PORTC_DATA;
// Set bit 0, port c as output.
    [p0]=r1;
// Endless loop.
pulser:
    [p1]=r0;
    [p1]=r1;
    jump pulser;
    rts;
._main.end:
```

```
/*
Program ASM inout demonstrates a simple audio in-out capability of the ADSP-BF706 EZ-KIT
Mini evaluation system using a polled,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 : 31.01.2017
*/

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

_main:
call codec_configure;
call sport_configure;
get_audio:
wait_left:
R0=[REG_SPORT0_CTL_B]; CC=BITTST(R0, 31); if !CC jump wait_left; // Wait left flag in
R0=[REG_SPORT0_RXPRI_B]; [REG_SPORT0_TXPRI_A]=R0; // Write left out
wait_right:
R0=[REG_SPORT0_CTL_B]; CC=BITTST(R0, 31); if !CC jump wait_right; // Wait right flag in
R0=[REG_SPORT0_RXPRI_B]; [REG_SPORT0_TXPRI_A]=R0; // Write right out
jump get_audio;
rts;
._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 26-59, 26-67,
// 26-75 and 26-76 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 24-15 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 ASM FIR 1 demonstrates a simple audio FIR filtering capability of the ADSP-BF706
EZ-KIT Mini evaluation system using a polled, sample-by-sample algorithm. The program is
written in assembly code, so is optimised for high speed operation. It is also entirely
self-contained, using only the header defBF706.h for the addresses of the BF706 egisters.
The version below can uses single MAC instructions, and can handle up to 8050 taps.

Author: Patrick Gaydecki
Date : 24.02.2017
*/

.section program;
.global _main;
.align 4;
#include <defBF706.h>
# define n 8
# define na2 0.125r

_main:
call codec_configure;
call sport_configure;
// Load filter coefficients in L1 memory, Block B. All are same value.
// The w[] operator does a 16-bit load.
P0=n;
R0=0;
R0.L=na2;
I1=0x11900000;
loop LC0=P0;
W[I1++]=R0.L;
loop_end;
I0=0x11800000;B0=I0;L0=(n*2); // Circular buffer for input data, Block A
I1=0x11900000;B1=I1;L1=(n*2); // Circular buffer for filter coefficients, Block B
P0=n-1;
get_audio:
wait_left:
R0=[REG_SPORT0_CTL_B]; CC=BITTST(R0, 31); if !CC jump wait_left; // Wait left flag in
R0=[REG_SPORT0_RXPRI_B];
A0 = 0 || W[I0++]= R0.H|| R1.H = W[I1++];
P0=n-1;
LOOP LC0=P0;
A0+= R0.H * R1.H|| R0.H = W[I0++] || R1.H = W[I1++]; // Filter left
LOOP_END;
A0+= R0.H * R1.H||I0-=2;
R0=A0; [REG_SPORT0_TXPRI_A]=R0;
wait_right:
R0=[REG_SPORT0_CTL_B]; CC=BITTST(R0, 30); if !CC jump wait_right; // Wait right flag in
R0=[REG_SPORT0_RXPRI_B]; [REG_SPORT0_TXPRI_A]=R0; // Write right out
jump get_audio;
rts;
._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 26-59, 26-67,
// 26-75 and 26-76 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 from 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 24-15 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 prescale 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 ASM FIR 2 demonstrates a simple audio FIR filtering capability of the ADSP-BF706
EZ-KIT Mini evaluation system using a polled, sample-by-sample algorithm. The program is
written in assembly code, so is optimised for high speed operation. It is also entirely
self-contained, using only the header defBF706.h for the addresses of the BF706
registers. The version below  uses dual MAC instructions, and can handle up to 12860
taps. It cannot process 16000 taps since the use of 2-byte boundary alignment (which
happens within the main loop on alternate cycles) requires 1.5 CPU cycles. The first null
point of this filter is 48000/12860 = 3.7325 Hz.

Author: Patrick Gaydecki
Date : 04.05.2017
*/
.section program;
.global _main;
# include <defBF706.h>
# define N 12860
# define NN2 6430
# define GAIN 0.0001r

_main:
P2=0;
call codec_configure;
call sport_configure;
// Load filter coefficients in L1 memory, Block B. All are same value.
// The w[] operator does a 16-bit load.
P0=16000;
R0=0;
R0.L=GAIN;
I1=0x11800000;
R2=N*2;
R3=I1;
R4=R2+R3;
I3=R4;
I1=0x11900000;
loop LC0=P0;
W[I1++]=R0.L;
loop_end;
I0=0x11800000;B0=I0;L0=N*2; // Circular buffer for input data, Block A
I1=0x11900000;B1=I1;L1=N*2; // Circular buffer for filter coefficients, Block B
P0=NN2-1;
get_audio:
wait_left:
R0=[REG_SPORT0_CTL_B]; CC=BITTST(R0, 31); if !CC jump wait_left; // Wait left flag in
R0=[REG_SPORT0_RXPRI_B];
A0 = 0 || W[I0]= R0.H || R1=[I1++];
R2=I0;
// Next three lines load a new word past the end point of the buffer due to addressing
// issues associated with dual MAC operations
CC=R2==R3;
IF !CC JUMP SK1;
W[I3]=R0.H;
SK1:
A1=0 || R0=[I0++];
//cycles=-4; // Diagnostic
LOOP LC0 = P0;
A0+= R0.L * R1.L, A1+= R0.H * R1.H || R0=[I0++] || R1=[I1++]; // Filter left
LOOP_END;
A0+= R0.L * R1.L, A1+= R0.H * R1.H || I0-=2;
A0+=A1;
R0=A0;
[REG_SPORT0_TXPRI_A]=R0;
wait_right:
R0=[REG_SPORT0_CTL_B]; CC=BITTST(R0, 30); if !CC jump wait_right; // Wait right flag in
R0=[REG_SPORT0_RXPRI_B]; [REG_SPORT0_TXPRI_A]=R0; // Write right out
jump get_audio;
rts;
._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 26-59, 26-67,
// 26-75 and 26-76 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 from 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 24-15 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 prescale 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 ASM FIR 3 is a single MAC FIR filtering program similar to ASM FIR 1. However, it
reads in a file of coefficients created by Signal Wizard. Also included is a commented-
out .BYTE buffer, which shows how coefficients may be hard-coded into the program..

Author: Patrick Gaydecki
Date  : 08.05.2017
*/

.section L1_data_b;  // Linker places data starting at 0x11900000
//.BYTE2 filter[]=0.1r,0.1r,0.1r,0.1r,0.1r,0.1r,0.1r,0.1r,0.1r,0.1r,0.1r,0.1r;
.BYTE2 filter[]="filter.txt"; // Load 16-bit coefficients from file
.section program;
.global _main;
.align 4;
# include <defBF706.h>

_main:
call codec_configure;
call sport_configure;
P0=length(filter)*2;
R0=0;
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=length(filter)-1;
get_audio:
wait_left:
R0=[REG_SPORT0_CTL_B]; CC=BITTST(R0, 31); if !CC jump wait_left; // Wait left flag in
R0=[REG_SPORT0_RXPRI_B];
A0 = 0 || W[I0++]= R0.H|| R1.H = W[I1++];
LOOP LC0=P0;
A0+= R0.H * R1.H|| R0.H = W[I0++] || R1.H = W[I1++]; // Filter left
LOOP_END;
A0+= R0.H * R1.H||I0-=2;
R0=A0; [REG_SPORT0_TXPRI_A]=R0;
wait_right:
R0=[REG_SPORT0_CTL_B]; CC=BITTST(R0, 30); if !CC jump wait_right; // Wait right flag in
R0=[REG_SPORT0_RXPRI_B]; [REG_SPORT0_TXPRI_A]=R0; // Write right out
jump get_audio;
rts;
._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 26-59, 26-67,
// 26-75 and 26-76 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 from 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 24-15 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 prescale 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 ASM FIR 4 is a dual MAC FIR filter similar to ASM FIR 2. Here, it  imports the
filter coefficients as a file. Tests confirm it can handle a maximum of 12860
coefficients.

Author: Patrick Gaydecki
Date : 08.05.2017
*/
.section L1_data_b;           // Linker places data starting at 0x11900000
.BYTE2 filter[]="filter.txt"; // Load 16-bit coefficients from file
.section program;
.global _main;
.align 4;
# include <defBF706.h>

_main:
call codec_configure;
call sport_configure;
// Set up various indices for addressing.
P2=-1;
P0=length(filter)*2;
P1=P0>>2;P1=P1+P2;
I1=0x11800000;
R2=P0;
R3=I1;
R4=R2+R3;
I3=R4;
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
get_audio:
wait_left:
R0=[REG_SPORT0_CTL_B]; CC=BITTST(R0, 31); if !CC jump wait_left; // Wait left flag in
R0=[REG_SPORT0_RXPRI_B];
A0 = 0 || W[I0]= R0.H || R1=[I1++];
R2=I0;
// load a new word past the end point of the buffer due to dual MAC addressing
CC=R2==R3; IF !CC JUMP SK1; W[I3]=R0.H;
SK1:
A1=0 || R0=[I0++];
LOOP LC0 = P1;
A0+= R0.L * R1.L, A1+= R0.H * R1.H || R0=[I0++] || R1=[I1++]; // Filter left
LOOP_END;
A0+= R0.L * R1.L, A1+= R0.H * R1.H || I0-=2;
A0+=A1;
R0=A0;
[REG_SPORT0_TXPRI_A]=R0;
wait_right:
R0=[REG_SPORT0_CTL_B]; CC=BITTST(R0, 30); if !CC jump wait_right; // Wait right flag in
R0=[REG_SPORT0_RXPRI_B]; [REG_SPORT0_TXPRI_A]=R0; // Write right out
jump get_audio;
rts;
._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 26-59, 26-67,
// 26-75 and 26-76 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 from 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 24-15 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 prescale 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 ASM FIR 5 is a single MAC FIR filtering program similar to ASM FIR 3. However,
this is a 32-bit version, requiring  combined A1:A0 72-bit accumulator register. Again, it
can handle up to 8050 coefficients, although a the available memory limits this to 7680.

Author: Patrick Gaydecki
Date  : 12.05.2017
*/

.section L1_data_b;  // Linker places data starting at 0x11900000
.byte4/r32 filter[]="filter.txt"; // Load 32-bit coefficients from file
.section program;
.global _main;
.align 4;
# include <defBF706.h>

_main:
call codec_configure;
call sport_configure;
P0=length(filter)*4;
I0=0x11800000;B0=I0;L0=P0; // Circular buffer for input data, Block A
I1=filter;B1=I1;L1=P0;     // Circular buffer for filter coefficients, Block B
P0=length(filter)-1;
get_audio:
wait_left:
R0=[REG_SPORT0_CTL_B]; CC=BITTST(R0, 31); if !CC jump wait_left; // Wait left flag in
R0=[REG_SPORT0_RXPRI_B];
A0 = 0 || [I0++]= R0 || R1 = [I1++];
A1=0;
LOOP LC0=P0;
A1:0+= R0 * R1 || R0 = [I0++] || R1 = [I1++]; // Filter left
LOOP_END;
A1:0+= R0*R1 || I0-=4;
R1=A1:0; [REG_SPORT0_TXPRI_A]=R1;
wait_right:
R0=[REG_SPORT0_CTL_B]; CC=BITTST(R0, 30); if !CC jump wait_right; // Wait right flag in
R0=[REG_SPORT0_RXPRI_B]; [REG_SPORT0_TXPRI_A]=R0; // Write right out
jump get_audio;
rts;
._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 26-59, 26-67,
// 26-75 and 26-76 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 from 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 24-15 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 prescale 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 ASM FIR 6 is a single MAC FIR filtering program similar to ASM FIR 5. However, it
initially loads the coefficients into L2 memory before moving them to L1. This is to
avoid the memory issues which limit ASM FIR 5.

Author: Patrick Gaydecki
Date  : 16.05.2017
*/

.section L2_sram;  // Linker places data into L2
.byte4/r32 filter[]="filter.txt"; // Load 32-bit coefficients from file
.section program;
.global _main;
.align 4;
# include <defBF706.h>

_main:
call codec_configure;
call sport_configure;
// Move the coefficients into L1.
P0=length(filter);
I0=filter;
I1=0x11900000;
LOOP LC0=P0;
R0=[I0++];
[I1++]=R0;
LOOP_END;
P0=length(filter)*4;
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=length(filter)-1;
get_audio:
wait_left:
R0=[REG_SPORT0_CTL_B]; CC=BITTST(R0, 31); if !CC jump wait_left; // Wait left flag in
R0=[REG_SPORT0_RXPRI_B];
A0 = 0 || [I0++]= R0 || R1 = [I1++];
A1=0;
LOOP LC0=P0;
A1:0+= R0 * R1 || R0 = [I0++] || R1 = [I1++]; // Filter left
LOOP_END;
A1:0+= R0*R1 || I0-=4;
R1=A1:0; [REG_SPORT0_TXPRI_A]=R1;
wait_right:
R0=[REG_SPORT0_CTL_B]; CC=BITTST(R0, 30); if !CC jump wait_right; // Wait right flag in
R0=[REG_SPORT0_RXPRI_B]; [REG_SPORT0_TXPRI_A]=R0; // Write right out
jump get_audio;
rts;
._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 26-59, 26-67,
// 26-75 and 26-76 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 from 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 24-15 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 prescale 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 ASMF7 is a single MAC FIR filtering program similar to ASM FIR 5. However, it
initially loads the coefficients into L2 memory before moving them to L1. This is to
avoid the memory issues which limit ASM FIR 5. The program can handle 8050 32-bit
coefficients. It alo sets the clock for maximum (boot mode).

Author: Patrick Gaydecki
Date  : 18.08.2017
*/

.section L2_sram;  // Linker places data into L2
.byte4/r32 filter[]="filter.txt"; // Load 32-bit coefficients from file
.section program;
.global _main;
.align 4;
# include <defBF706.h>
_main:
call codec_configure;
call sport_configure;
// Set the clocks to maximum
R1=0X2000;
[REG_CGU0_CTL]=R1;
R1=0X42042442;
[REG_CGU0_DIV]=R1;
// Move the coefficients into L1
P0=length(filter);
I0=filter;
I1=0x11900000;
LOOP LC0=P0;
R0=[I0++];
[I1++]=R0;
LOOP_END;
P0=length(filter)*4;
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=length(filter)-1;
r3.h=0.5r;
get_audio:
wait_left:
R0=[REG_SPORT0_CTL_B]; CC=BITTST(R0, 31); if !CC jump wait_left; // Wait left flag in
R0=[REG_SPORT0_RXPRI_B];
A0 = 0 || [I0++]= R0 || R1 = [I1++];
A1=0;
LOOP LC0=P0;
A1:0+= R0 * R1 || R0 = [I0++] || R1 = [I1++]; // Filter left
LOOP_END;
A1:0+= R0*R1 || I0-=4;
R1=A1:0; r1=r1*r3;
[REG_SPORT0_TXPRI_A]=R1;
wait_right:
R0=[REG_SPORT0_CTL_B]; CC=BITTST(R0, 30); if !CC jump wait_right; // Wait right flag in
R0=[REG_SPORT0_RXPRI_B]; [REG_SPORT0_TXPRI_A]=R0; // Write right out
jump get_audio;
rts;
._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 26-59, 26-67,
// 26-75 and 26-76 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 from 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 24-15 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 prescale 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:
```

To assemble, link, produce a loader file and finally program the flash in boot mode (SPI master) on the ADSP-BF706 EZ-KIT without CCES, use the following (minimal) command sequence:

```
easmblkfn -proc ADSP-BF706 mycode.asm

ccblkfn.exe -proc ADSP-BF706 mycode.doj -o mycode.dxe

elfloader.exe -v -proc ADSP-BF706 -si-revision 1.0 -b SPI -f hex -width 8 -bcode 1 -o
mycode.ldr mycode.dxe

cldp -proc ADSP-BF706 -emu kit -driver bf706_w25q32bv_dpia.dxe -cmd prog -file mycode.ldr
```

Alternatively, use CCES to produce the .ldr file and simply invoke the final command above. Follow these steps:

- Go to the "Project" menu and choose "Properties", then "C/C++ Build", then "Settings", then "Build Artifact" tab.
- Go to "Artifact Type". From the dropdown menu select "Loader File".
- Click on the "Tool Settings" tab and select "CrossCore Blackfin Loader" then "General".
- For "Boot mode" choose "SPIO master" for serial.
- For "Boot format" choose "Intel hex".
- For "Output width" choose "8-bits".
- For "Boot code" enter 1".
- "Initialization file" should not be needed unless part of the example is being loaded to external memory.
- "Use default start address" should be checked.
- Click "OK".
- Rebuild the project to generate the LDR file.