# ELEN90058 Signal Processing

## Workshop 3

## Implementation of linear filters

**Aim:** 1) To implement an efficient overlap add algorithm for filtering of real signals in Matlab 2) To design a digital filter using Matlab and to implement it using both time domain and FFT based methods on an Analog Devices BF533 DSP board.

**Lab:** The workshop consists of assignment type questions (Part A), a component on Matlab implementation of the overlap add algorithm (Part B) and a hardware component on filter implementation in real time using Analog Devices BF533 DSP. Note that Part A and B can (and should) be done outside the workshop hours, so that you can focus on the DSP part in the workshop hours. **In particular, design and test the filter in Matlab task 4a) before your first workshop.**

**Assessment:** The workshop is assessed based on a written workshop report and on demonstrations of the algorithms implemented on the DSP board in the workshops. The workshop constitute 6% of the overall assessment of the course

**Submission of the reports:** The students are to submit the report to the demonstrator in the last week of the semester in your workshop time slot.. You also have to sign a declaration that the report is your own work.

**Student groups:** Students should work in groups of three.

**Reports:** The reports should cover all parts, i.e. Parts A, B and C. The reports should be clearly written and explain in a logical way how the different tasks in the lab have been carried out. The designs and choices you make (e.g. filter order, cut off frequencies, allowed ripples, etc.) should be explained and justified. The Matlab code must be easy to read and well documented. Results obtained using Matlab must be explained, i.e. it is not sufficient to copy the output of Matlab without further explanations of what the numbers or graphs mean. Figures should be included where it is appropriate. The Matlab and c code should be included in an appendix. See also the note on LMS about workshop expectations and reports. The maximum length of the reports are 20 pages excluding the appendix.

**Collaboration between and within groups:** It is perfectly OK to discuss problems and possible solutions with other groups. However, each group has to carry out the lab independently, and e.g. copying of other groups' mathematical derivations, c code or Matlab code is not acceptable. Both group members should do an equal amount of work on both the lab itself and the writing of the report. It is not acceptable that one group member does the lab and the other writes up the results.

## 1 Part A: Assignment type questions

In this part we will look at the theory behind implementations of the overlap add method for filtering of real data sequences.

**Question 1**

**a)** Briefly explain how the overlap add method works.

**Question 2**

In the output stage of FFT based methods you need to calculate an Inverse Discrete Fourier Transform (IDFT). We will now look at efficient algorithms for implementing the IDFT using the FFT. Chapter 11.3.3 (3rd ed), 11.3.4 (4th ed) in Mitra outlines one approach. Another approach is explored in the next question.

**a)** Let $x[n]$ be a length-$N$ sequence and consider the following $N$-point DFTs.

$$X[k] = \text{DFT}\{x[n]\}$$

and

$$y[n] = \text{DFT}\{X[k]\}$$

i.e. $y[n]$ is the DFT of the DFT of $x[n]$. Express $y[n]$ in terms of $x[n]$ and explain how you can use this results to implement the IDFT algorithm.

**Question 3**

In Chapter 5.9.2 in Mitra it is explained how you can compute a $2N$-point DFT of a real sequence using a single $N$-point DFT. We will now look at how we can compute a $2N$-point IDFT of a conjugate symmetric sequence using a single $N$-point IDFT.

**a)** Show that if $X[k], k = 0, \dots, 2N-1$ is conjugate symmetric, i.e.

$$X[k] = X^*[\langle -k \rangle_{2N}]$$

then $x[n] = \text{IDFT}\{X[k]\}$ is real.

**b)** Let

$$\begin{aligned} X_0[k] &= X[k] + X[k+N], \quad k = 0, \dots N-1 \\ X_1[k] &= W_{2N}^{-k}(X[k] - X[k+N]), \quad k = 0, \dots N-1 \end{aligned}$$

Show that the length-$N$ sequence $X_0[k]$ is conjugate symmetric ($X_0[k] = X_0^*[\langle -k \rangle_N]$) and that the length-$N$ sequence $jX_1[k]$ is conjugate anti-symmetric ($jX_1[k] = -(jX_1[\langle -k \rangle_N])^*$).

**c)** Let $Q[k] = X_0[k] + jX_1[k]$. Let the $N$-point IDFT of $Q[k]$ be $q[n]$. Show that

$$\begin{aligned} x[2n] &= \frac{1}{2}\text{Re}\{q[n]\}, \quad n = 0, \dots N-1 \\ x[2n+1] &= \frac{1}{2}\text{Im}\{q[n]\}, \quad n = 0, \dots N-1 \end{aligned}$$

**d)** Explain how you can use the above results to compute a $2N$ point IDFT of a conjugate symmetric sequence using a single $N$-point DFT.

**e)** An implementation of the overlap add method for real signals has made use of the approach in Chapter 5.9.2 in Mitra for the calculation of the FFT of a real sequence and the approach above for the calculation of the IFFT of a conjugate symmetric sequence. Assume that the input sequence is very long. Using an analysis similar to Chapter 6.2.3 3rd Ed (Chapter 8.2.3, 4th Ed) in Proakis and Manolakis (and also sketched in the lecture notes on pages XIV-17,18), calculate how many complex multiplications have to be performed per output point for a given filter of order $M$ when the block size is $N$.

## 2 Part B: Implementation of the overlap add method for real signals in Matlab

In this part you are going to implement the overlap add method for real signals making use of the theory developed in Part A.

**Common Matlab mistake.** `x'` is the complex conjugate transpose of the matrix or vector `x`. It is **not** the transpose for complex `x`.

$\boxed{\textbf{Task 1}}$

**a)** Write two Matlab functions `iffta(X)` and `ifftb(X)` which implement the algorithm in Chaper 11.3.3 (3rd ed), 11.3.4 (4th ed) in Mitra and question **2a)**. You can make use of the `fft` function in Matlab.

**b)** Generate a complex test signal of length 16 using the `randn` command in Matlab and verify that the functions you have written gives the same output as the inbuilt `ifft` function in Matlab.

$\boxed{\textbf{Task 2}}$

**a)** Write a Matlab function which test whether a sequence is conjugate symmetric. Allow for round off errors so that the match does not have to be exact.

**b)** Write a Matlab function `ifftcs(X)` which compute the IFFT of a conjugate symmetric sequence of length $2N$ using the approach in question **3)**. Return an error message if the sequence is not conjugate symmetric.

**c)** Generate a conjugate symmetric test signal of length 16 and verify that your function `ifftcs` returns the same result as `ifft`.

$\boxed{\textbf{Task 3}}$

**a)** Write a Matlab function `overlapaddreal(B,x,N)` which filters the real sequence `x` with the FIR filter whose impulse response coefficients are stored in the vector `B` using the overlap add method. `N` is the block length.

**Task 4**

**a)** The sampling frequency is 24kHz. Design an FIR filter according to the following specifications

| | |
|---|---|
| Passband | $[0, 220 \text{ Hz}]$ |
| Stopband | Above 880 Hz |
| Passband ripple | 0.05 |
| Stopband ripple | 0.05 |

**b)** Generate a suitable test signal and implement the filter in **a)** using the Matlab routine you developed in Task 3. What is the optimal block length? Verify that your routine gives the same output as the `filter` and `fftfilt` command in Matlab. Explain briefly how the Matlab commands `filter` and `fftfilt` implements a linear filter.

**c)** How many complex multiplications and additions per second are required to implement the filter in each case? Include savings due to any symmetries. *Hint:* It may be easier to work out how many calculations are required per sampling period or output block first.

## 3 Part C: Real-time implementation of the DFT-based filtering methods on DSP hardware

In this part you will be using the Analog Devices BF533 EZ-Kit Lite board as in previous workshops. In this lab, the sampling frequency is 24 kHz. The tasks involve completing functions in the `Lab_Tasks.c` file. Some programming hints are given in the appendix which you should read before commencing the tasks.

### 3.1 Hardware setup (same as Workshop 2)

The ADSP-BF533 evaluation board has many features, but for this workshop you will be using only a subset of the components. The AD1836 codec is responsible for acquiring the data from the Stereo IN Phono Jack and passing it to the ADSP-BF533 processor via the SPORT0 buffer. After adjustments are made to the signal (it is up to you to program these adjustments) the data is passed back to the AD1836 codec and output onto the Stereo OUT Phono Jack, connected to some speakers.

Before you begin this part of the workshop, make sure that the cabling is correct. You should have power on the evaluation board, a USB connection to the computer,

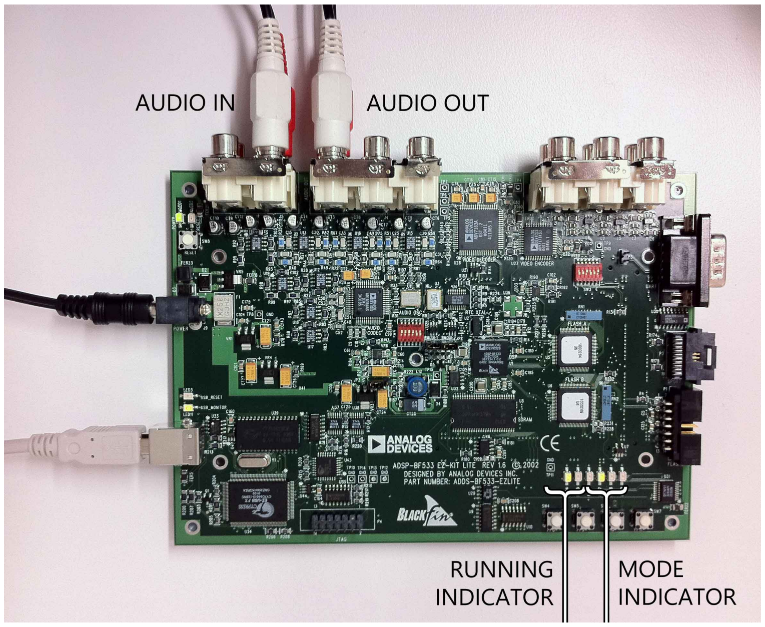


Figure 1: Hardware setup

sound going into AUDIO IN and speakers connected to AUDIO OUT. If you are unsure of any of these, see your workshop demonstrator.

**WARNING** Never put on a headset or have the loudspeakers turned up at high volume before you have verified that the code is working as expected. If a filter is unstable, very loud and unpleasant audio output is generated.

## 3.2 Tasks

**a)** As in previous workshops, a project containing the skeleton code is available on LMS. Start CCES and verify the skeleton project is working as intended. You should hear continuous sound playing from your source.

**b)** Import the impulse response coefficients (see Appendix B)of the filter you designed in the Matlab part into the `Lab_Tasks.c` file. You will also need to modify the `Params.h` file to specify the filter length $M$ :

```
#define M 2
```

**c)** Implement the filter you have designed in time domain using similar methods as previous workshops. This involves completing the `process_time` function in the `Lab_Tasks.c` file.

Because the order of the filter may be high, you should take advantage of any symmetry in the coefficients to reduce the number of floating point multiplications required.

**d)** You will now implement the filter using the overlap add method.

Using an FFT based method such as overlap add to process data online involves acquiring blocks of data, processing them and outputting the result. For your convenience, the process of acquiring blocks of data has already been coded and you only need to consider the processing functions.

The header file contains a `#define` for the length of the FFT $N$. Let $N$ be the optimal block length you found in Task 4 in the Matlab part.

```
#define N 128
```

The `init_process()` function (provided) initialises the FFT twiddle factors and computes the FFT of the filter coefficients. Take a look. See Appendix A for information about the `fract32` and `complex_fract32` data types. The `rfft_fr32` function calculates the FFT of a real-valued array.

```
void rfft_fr32(const fract32                   input[],
               complex_fract32                 output[],
               const complex_fract32           twiddle_table[],
               int                             twiddle_stride,
               int                             fft_size,
               int                            *block_exponent,
               int                             scale_method);
```

The `twiddle_stride` parameter is used for efficiency when you wish to calculate multiple FFTs of different lengths so that a single twiddle table can be used. The `block_exponent` parameter is an output which gives a $2^{\nu}$ scaling factor. The `scale_method` used should be `1`. After processing, you will need to rescale the result by this amount (see the following example).

```
int blk_exp;
rfft_fr32(input, output, twiddle, 1, N, &blk_exp, 1);

// ...

// rescale
for (i = 0; i < N; i++)
    output[i] = output[i] << blk_exp;
```

Implement the filter you have designed using the overlap-add method. The buffering of the input and output data has already been implemented for you in the `process_data.c` file (take a look to see what steps are involved). The `process_block()` function is called once the input buffer is full.

The input data is stored in the `fract32[]` array `input_data`. You are required to compute the output data and store it in the array `output`. For simplicity, we will only process the left audio channel.

The `ifft_fr32` function computes the inverse FFT, or you may use one of the methods in the preparation section to efficiently compute it.

```
void ifft_fr32(const complex_fract32     input[],
               complex_fract32           output[],
               const complex_fract32     twiddle_table[],
               int                       twiddle_stride,
               int                       fft_size,
               int                      *block_exponent,
               int                       scale_method);
```

The immediate result of your calculations will be of a complex datatype, but the output will be real so the final stage of your processing should be to copy just the real part to the `output_data` array.

**Show and demonstrate your filter implementations to your workshop demonstrator.**

## A Programming hints

### A.1 Fixed point and complex number arithmetic

The BF533 is a fixed point processor, and floating point multiplications programmed in C are time consuming. Due to the high number of operations that are required to implement FFTs, in this lab you will be programming the BF533 using fixed point (integer) arithmetic. Despite the prevalence of powerful floating point processors, many applications still prefer fixed point processing due to lower cost and size.

Because you will now need to use floating point and complex numbers, you will need to use the `complex_fract32` (32-bit complex fractional) datatype instead of `float`. Since this is not a primitive datatype in C, you will need to use the `cadd_fr32`, `csub_fr32`, `cmlt_fr32` and `cdiv_fr32` functions to perform arithmetic operations instead of the usual +, −, *, / symbols. For example, instead of

```
float a, b, c;
float d = (a + b) * c;
```

you should use the following commands instead:

```
complex_fract32 a, b, c;
complex_fract32 d = cmlt_fr32(cadd_fr32(a, b), c);
```

To obtain the real and imaginary parts of a complex number, you can access the `.re` and `.im` fields as follows.

```
complex_fract32 complex_number;
fract32 real_part = complex_number.re;
fract32 imaginary_part = complex_number.im;
```

For more information, refer to the *CCES Documentation for Blackfin* help file. In particular, the DSP Run-Time Library Reference may be useful.

## B Importing Matlab filter coefficients into CCES

In order to avoid having to type in large arrays of values into CCES a MATLAB function `print_array()` has been provided that converts a MATLAB array to a c array, which you can then copy and paste into your code.

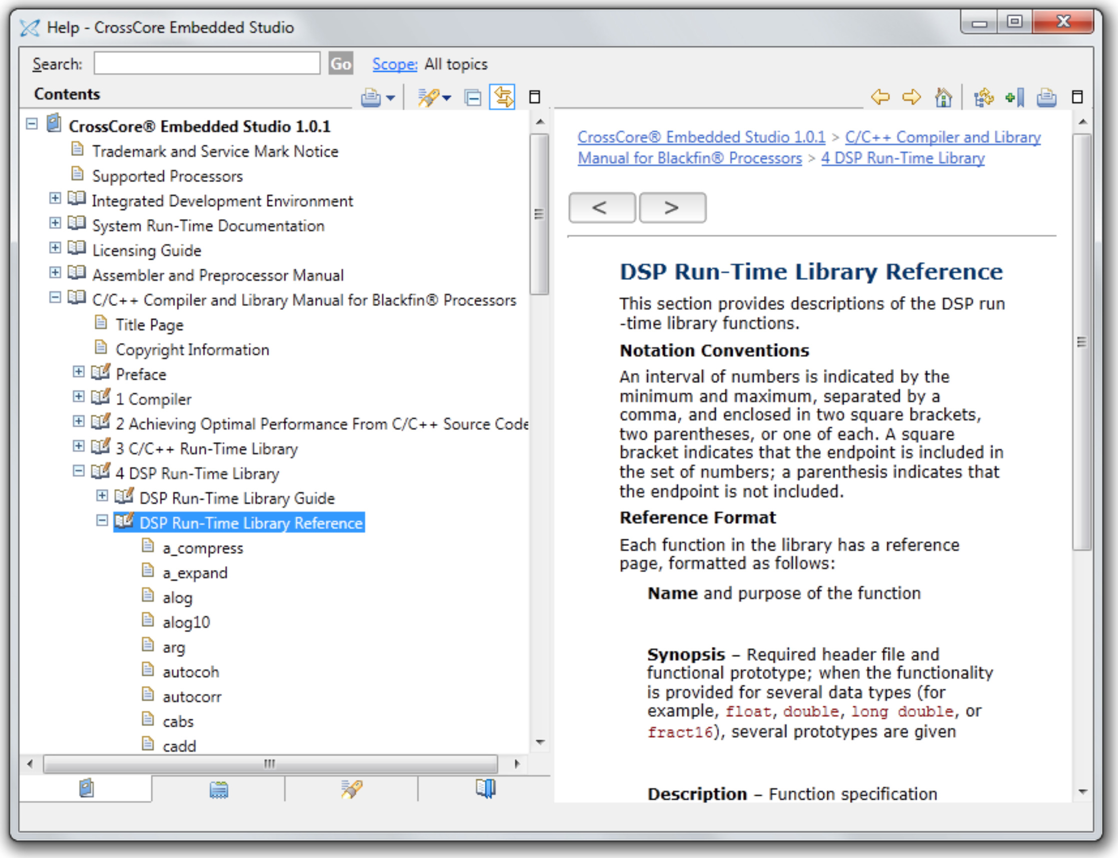
`print_array(array,name)` takes in a numeric array and prints a c array to the file array.txt as well as to the command prompt. The name of the array is given by `name`. If no name is provided the default name is 'A'.

For example, `print_array([1.2 3 2.4], 'x')` will return:

```
    // array x
float x[] = { 1.200000e+000, 3, 2.400000e+000 };
```

whereas `print_array([1.2 3 2.4])` returns:

```
    // array A
float A[] = { 1.200000e+000, 3, 2.400000e+000 };
```



CrossCore® Embedded Studio 1.0.1
- Trademark and Service Mark Notice
- Supported Processors
- Integrated Development Environment
- System Run-Time Documentation
- Licensing Guide
- Assembler and Preprocessor Manual
- C/C++ Compiler and Library Manual for Blackfin® Processors
  - Title Page
  - Copyright Information
  - Preface
  - 1 Compiler
  - 2 Achieving Optimal Performance From C/C++ Source Code
  - 3 C/C++ Run-Time Library
  - 4 DSP Run-Time Library
    - DSP Run-Time Library Guide
    - DSP Run-Time Library Reference
      - a_compress
      - a_expand
      - alog
      - alog10
      - arg
      - autocoh
      - autocorr
      - cabs
      - cadd

CrossCore® Embedded Studio 1.0.1 > C/C++ Compiler and Library Manual for Blackfin® Processors > 4 DSP Run-Time Library

DSP Run-Time Library Reference

This section provides descriptions of the DSP run-time library functions.

**Notation Conventions**

An interval of numbers is indicated by the minimum and maximum, separated by a comma, and enclosed in two square brackets, two parentheses, or one of each. A square bracket indicates that the endpoint is included in the set of numbers; a parenthesis indicates that the endpoint is not included.

**Reference Format**

Each function in the library has a reference page, formatted as follows:

**Name** and purpose of the function

**Synopsis** – Required header file and functional prototype; when the functionality is provided for several data types (for example, `float`, `double`, `long double`, or `fract16`), several prototypes are given

**Description** – Function specification

Figure 2: DSP Run-Time Library Reference in help files

## C Coding Hints

For those that are less familiar with C, here are some coding hints that may help you throughout the workshop:

- Global variables can be modified from anywhere in your project, and so you should be careful not to give your local variables identical names. Most variables that you will need have been defined as global rather than as function inputs and outputs for simplicity.
- Variables defined as `float` may lose resolution if multiplied or divided by a variable of type `int`. Any such multiplications should be typecast as floating point by typing `(float)` immediately before it.
- `Static` local variables are initialised only once when a function is first called, and retain their value upon subsequent calls to the same function. For this reason, they are useful for "remembering" the state of your program between events, without the need for using global variables (the advantage being that these can only be accessed within the local host function, and so will not be confused with other static variables of the same name in other functions). They are useful for interrupt driven applications, such as this one.
- In c an array with `N` elements is indexed from `0` to `N-1`.

## D Approximate weighting (Total: 6 marks)

| | |
|---|---|
| Part A: Assignment type questions: | 2.5 |
| Part B: Matlab part: | 2.0 |
| Part C: Hardware part: | 1.5 |