ULBS
Universitatea “Lucian Blaga” din Sibiu

Facultatea de Inginerie Hermann Oberth
Master-Program “Embedded Systems“
Advanced Digital Signal Processing Methods
Winter Semester 2014/2015

Report 4

Implementation of FIR filter

Professor: Prof. dr. ing. Mihu Ioan

Masterand: Stefan Feilmeier

23.01.2015

TABLE OF CONTENTS

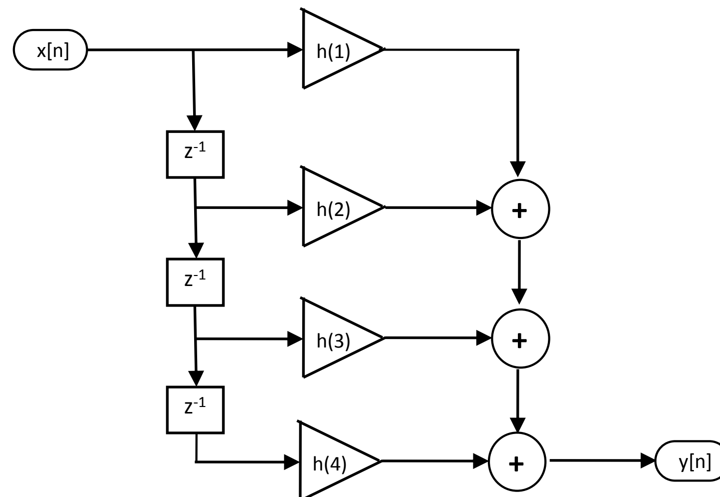
1 Task 3
2 Offline Processing 4
3 Online Processing 4
4 Source Code 6

1 TASK

This paper shows the implementation of a **non-recursive FIR (Finite Impulse Response) filter** in C programming language. The filter is defined by the equation

$$y_{[n]} = \sum_{k=0}^{N-1} h_{[k]} \times x_{[n-k]}$$

and is implemented in **direct-form FIR structure**:



As an example signal, an audio file of the song „Shift ft. Marius Moga - Sus Pe Toc” is used. It has a sampling **frequeny of 44.1000 Hz** and a total of **8.586.479 samples** in data type double.

The applied filter is a Low-Pass-Filter (LPF) with a **normalized cutoff frequency of 0.1** (2.205 Hz in our case) and 50 coefficients.

Two general methods will be implemented: online and offline processing.

2 OFFLINE PROCESSING

In “offline processing”, the data is already available before executing the filter.

The main part of the code is as follows:

```
/* Direct-form FIR structure
 *
 * x : input signal array
 * x_cnt : length of input signal array
 * h : coefficients array
 * h_cnt : length of coefficients array
 * y : output signal array
 */
static void offline_dffir(const double *x, unsigned long x_cnt,
        const double *h, unsigned int h_cnt, double *y) {
        unsigned long n;
        unsigned int k;
        for(n = h_cnt; n < x_cnt; n++) {
                y[n] = 0;
                for(k = 0; k < h_cnt; k++) {
                        y[n] += h[k] * x[n-k];
                }
        }
}
```

The pointer to the array holds the output signal.

3 ONLINE PROCESSING

In “online processing”, the data is handled as it arrives on an input interface, usually an Analogue Digital Converter (ADC). The actual processing of the FIR filter is fulfilled in the ADC interrupt routine.

The following code simulates an ADC converter with a two byte output. It therefor uses the input from the example file, quantifies it into a two byte short integer. The actual processing is done in a timer interrupt routine.

The result is printed on standard output.

```
/* Online direct-form FIR structure
 *
 * Interrupt handler
 * (this simulates an A/D converter)
 */
short *g_x;
unsigned long g_x_cnt;
void interrupt_handler(int sig) {
        static unsigned long n = 0;
        unsigned int k;
        double y = 0;
        // for safety: if we reach the end of the array
        if(n > g_x_cnt) return;
        // FIR
        for(k = 0; k < g_h_cnt && k < n; k++) {
                y += g_h[k] * g_x[n-k];
        }
        // Result
        printf("y=%f\n", y);
        n++;
}

/*
 *x : input signal array
 * x_cnt : length of input signal array
 *
 * Expected global variables:
 * g_h : coefficients array
 * g_h_cnt : length of coefficients array
 */
static void online_dffir(double *x, unsigned long x_cnt) {
        /* Quantify x as 2 byte integer - like from ADC and make publically
available */
        static short x_int[BUFFER_LEN];
        unsigned long i;
        g_x_cnt = x_cnt;
        for(i = 0; i < x_cnt; i++) {
                x_int[i] = x[i] * 32768;
        }
        g_x = x_int;
        /* Prepare timer interrupt */
        struct sigaction sa;
        struct itimerval timer;
        memset (&sa, 0, sizeof (sa));
        sa.sa_handler = &interrupt_handler;
        sigaction (SIGVTALRM, &sa, NULL);
        timer.it_value.tv_sec = 0;
        timer.it_value.tv_usec = 100000;
        timer.it_interval.tv_sec = 0;
        timer.it_interval.tv_usec = 100000;
        setitimer (ITIMER_VIRTUAL, &timer, NULL);
        /* Forever loop. */
        while(1);
        //sleep(60);
}
```

4 Source Code

The complete source code of the application also shows the reading and writing of the audio file using “libsndfile”[1], as well as additional implementations of the FIR filter in “Symmetric direct-form” and a version of “Symmetric direct-form” using fixed-point arithmetic using “fixedptc”[2].

fir.c

```
/* (c) Stefan Feilmeier, 2015
 *
 * Names:
 *   x = input signal
 *   y = output signal
 *   n = current sample
 *   h = filter coefficients
 *   k = current coefficient
 *   *_cnt = length of array
 */

#include <stdio.h>
#include <string.h>
#include <time.h>
#include <signal.h>
#include <sys/time.h>
#include <unistd.h>
#include <sndfile.h>
/* for simplicity: read complete file at once */
#define BUFFER_LEN 8586479
#define MAX_CHANNELS 1
#include "coeff.h"
extern const double g_h[];
extern const unsigned int g_h_cnt;
#define FIXEDPT_BITS 32
#define FIXEDPT_WBITS 4
#include "fixedptc.h"

/* Function declarations
 */
static int read_wav(double *x, unsigned long *x_cnt, SF_INFO *sfinfo);
static int write_wav(const double *y, unsigned long y_cnt, SF_INFO *sfinfo);
static void offline_dffir(const double *x, unsigned long x_cnt,
    const double *h, unsigned int h_cnt, double *y);
static void offline_dfsymfir(const double *x, unsigned long x_cnt,
    const double *h, unsigned int h_cnt, double *y);
static void offline_dfsymfir_fp(const fixedpt *xfp, unsigned long x_cnt,
    const fixedpt *hfp, unsigned int h_cnt, fixedpt *yfp);
static void online_dffir(double *x, unsigned long x_cnt);
```

[1] libsndfile is a C library for reading and writing files containing sampled sound (such as MS Windows WAV and the Apple/SGI AIFF format) through one standard library interface. It is released in source code format under the Gnu Lesser General Public License.
http://www.mega-nerd.com/libsndfile/

[2] fixedptc library - a simple fixed point math header library for C.
Copyright (c) 2010-2012. Ivan Voras <ivoras@freebsd.org>
Released under the BSDL.
http://sourceforge.net/projects/fixedptc/

```
static void convert_dbl_to_fp(const double *x, fixedpt *xfp, unsigned long
cnt);
void interrupt_handler(int sig);
//static void convert_fp_to_dbl(const fixedpt *xpt, double *x, unsigned long
cnt);

/* for debugging fixed point numbers */
void fixedpt_print(fixedpt A) {
    char num[20];
    fixedpt_str(A, num, -2);
    puts(num);
}

/* MAIN
 */
int main(int argc, char *argv[]) {
    // for time measurement
    time_t start, end;
    double dif;
    // for input data handling
    static double x[BUFFER_LEN], y[BUFFER_LEN];
    static unsigned long x_cnt;
    static SF_INFO sfinfo;
    memset (&sfinfo, 0, sizeof (sfinfo));

    // read wav file
    if( read_wav(x, &x_cnt, &sfinfo) != 0) return 1;

    time(&start);
    // evaluate parameters
    if(argc == 1) {
        printf("Please select filter\n");
        printf("  dffir: Offline direct-form FIR\n");
        printf("  dffir_online: Online direct-form FIR\n");
        printf("  dfsym: Offline Symmetric direct-form FIR\n");
        printf("  dfsym_fp: Offline Symmetric direct-form FIR with fixed-
point arithmetic\n");
        return 0;
        } else if (strcmp(argv[1], "dffir") == 0) {
        printf("Offline direct-form FIR structure\n");
        offline_dffir(x, x_cnt, g_h, g_h_cnt, y);
    } else if (strcmp(argv[1], "dffir_online") == 0) {
        printf("Online direct-form FIR structure\n");
        online_dffir(x, x_cnt);
    } else if (strcmp(argv[1], "dfsym") == 0) {
        printf("Offline Symmetric direct-form FIR structure\n");
        offline_dfsymfir(x, x_cnt, g_h, g_h_cnt, y);
    } else if (strcmp(argv[1], "dfsym_fp") == 0) {
        printf("Offline Symmetric direct-form FIR structure with fixed-point
arithmetic\n");
        static fixedpt xfp[BUFFER_LEN], yfp[BUFFER_LEN];
        static fixedpt hfp[50];
        convert_dbl_to_fp(x, xfp, x_cnt);
        convert_dbl_to_fp(g_h, hfp, g_h_cnt);
        offline_dfsymfir_fp(xfp, x_cnt, hfp, g_h_cnt, yfp);
    }

    // time measurement
    time(&end);
    dif = difftime(end, start);
```

```
    printf("Filtered in %.2lf seconds\n", dif);

    // write output file
    if( write_wav(y, x_cnt, &sfinfo) != 0) return 1;

    return 0;
}

/* Read WAV file
 */
static int read_wav(double *x, unsigned long *x_cnt, SF_INFO *sfinfo) {
    SNDFILE *infile;
    const char *infilename = "input.wav";
    //const char *infilename = "input-long.wav" ;

    if (! (infile = sf_open (infilename, SFM_READ, sfinfo))) {
        /* Open failed so print an error message. */
        printf ("Not able to open input file %s.\n", infilename);
        /* Print the error message from libsndfile. */
        puts (sf_strerror (NULL)) ;
        return 1 ;
    }
    if (sfinfo->channels > MAX_CHANNELS) {
        printf ("Not able to process more than %d channels\n", MAX_CHANNELS);
        return 1 ;
    }
    *x_cnt = sf_read_double (infile, x, BUFFER_LEN);
    sf_close (infile);
    return 0;
}

/* Write WAV file
 */
static int write_wav(const double *y, unsigned long y_cnt, SF_INFO *sfinfo) {
    SNDFILE *outfile;
    const char *outfilename = "output.wav";

    if(! (outfile = sf_open (outfilename, SFM_WRITE, sfinfo))) {
        printf ("Not able to open output file %s.\n", outfilename);
        puts (sf_strerror (NULL));
        return 1;
    }

    sf_write_double(outfile, y, y_cnt);
    sf_close (outfile);
    return 0;
}

/* Direct-form FIR structure
 */
static void offline_dffir(const double *x, unsigned long x_cnt,
    const double *h, unsigned int h_cnt, double *y) {
    unsigned long n;
    unsigned int k;
    for(n = h_cnt; n < x_cnt; n++) {
        y[n] = 0;
        for(k = 0; k < h_cnt; k++) {
            y[n] += h[k] * x[n-k];
        }
    }
```

```
}

/* Symmetric direct-form FIR structure
 */
static void offline_dfsymfir(const double *x, unsigned long x_cnt,
    const double *h, unsigned int h_cnt, double *y) {
    unsigned long n;
    unsigned int k;
    unsigned int h_cnt2 = h_cnt / 2;
    unsigned int k_rev;
    for(n = h_cnt; n < x_cnt; n++) {
        y[n] = 0;
        for(k = 0; k < h_cnt2; k++) {
            k_rev = h_cnt - k;
            y[n] += h[k] * x[n-k] + h[k_rev] * x[n-k_rev];
        }
    }
}

/* Offline Symmetric direct-form FIR structure with fixed-point arithmetic
 */
static void offline_dfsymfir_fp(const fixedpt *xfp, unsigned long x_cnt,
        const fixedpt *hfp, unsigned int h_cnt, fixedpt *yfp) {
    unsigned long n;
    unsigned int k;
    unsigned int h_cnt2 = h_cnt / 2;
    unsigned int k_rev;
    for(n = h_cnt; n < x_cnt; n++) {
        yfp[n] = fixedpt_fromint(0);
        for(k = 0; k < h_cnt2; k++) {
            k_rev = h_cnt - k;
            yfp[n] = fixedpt_add( yfp[n], fixedpt_add(
                fixedpt_xmul( hfp[k], xfp[n-k] ),
                fixedpt_xmul( hfp[k_rev], xfp[n-k_rev] )
            ));
        }
    }
}

/* Online direct-form FIR structure
 *
 * Interrupt handler
 * (this simulates an A/D converter)
 */
short *g_x;
unsigned long g_x_cnt;
void interrupt_handler(int sig) {
    static unsigned long n = 0;
        unsigned int k;
    double y = 0;
    // for safety: if we reach the end of the array
    if(n > g_x_cnt) return;
    // FIR
    for(k = 0; k < g_h_cnt && k < n; k++) {
        y += g_h[k] * g_x[n-k];
    }
    // Result
    printf("y=%f\n", y);
    n++;
}
```

```
static void online_dffir(double *x, unsigned long x_cnt) {
    /* Quantify x as 2 byte integer - like from ADC and make publically
available */
    static short x_int[BUFFER_LEN];
    unsigned long i;
    g_x_cnt = x_cnt;
    for(i = 0; i < x_cnt; i++) {
        x_int[i] = x[i] * 32768;
    }
    g_x = x_int;
    /* Prepare timer interrupt */
    struct sigaction sa;
    struct itimerval timer;
    memset (&sa, 0, sizeof (sa));
    sa.sa_handler = &interrupt_handler;
    sigaction (SIGVTALRM, &sa, NULL);
    timer.it_value.tv_sec = 0;
    timer.it_value.tv_usec = 100000;
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 100000;
    setitimer (ITIMER_VIRTUAL, &timer, NULL);
    /* Forever loop. */
    while(1);
    //sleep(60);
}

/* Convert double array to fixed-point array
 */
static void convert_dbl_to_fp(const double *x, fixedpt *xfp,
unsigned long cnt) {
    unsigned long n;
    for(n = 0; n < cnt; n++) {
        xfp[n] = fixedpt_rconst(x[n]);
    }
}

/* Convert fixe-point array to double array
 */
/*static void convert_fp_to_dbl(const fixedpt *xfp, double *x, unsigned long
cnt) {
    unsigned long n;
    for(n = 0; n < cnt; n++) {
        x[n] = fixedpt_rconst(x[n]);
    }
}*/
```

coeff.h

```
/* (c) Stefan Feilmeier, 2015
 */
const double g_h[] = { 0.001016, 0.0010522, 0.0010549, 0.00095267,
0.00063961, -6.5267e-019, -0.0010569, -0.0025587, -0.0044351, -0.006495, -
0.0084214, -0.0097882, -0.010099, -0.0088474, -0.0055854, 2.6524e-018,
0.0080221, 0.018347, 0.030575, 0.044053, 0.057923, 0.071196, 0.082857,
0.091965, 0.097759, 0.099748, 0.097759, 0.091965, 0.082857, 0.071196,
0.057923, 0.044053, 0.030575, 0.018347, 0.0080221, 2.6524e-018, -0.0055854, -
0.0088474, -0.010099, -0.0097882, -0.0084214, -0.006495, -0.0044351, -
0.0025587, -0.0010569, -6.5267e-019, 0.00063961, 0.00095267, 0.0010549,
0.0010522, 0.001016 };

const unsigned int g_h_cnt = sizeof(g_h)/sizeof(g_h[0]);
```