AFEKA TEL-AVIV ACADEMIC COLLEGE OF ENGINEERING

פרויקט סיום קורס:

20210 יישומי מחשב להנדסת אלקטרוניקה

שם הפרויקט:

"נתח תדרים ואוסצילוסקופ עבור אותות שמע"

מגיש: מוטי חדש

ת"ז:

שם המרצה: פרופסור אלי פלקסר

תאריך הגשה: 15.02.18

מהות הפרויקט:

בעולם השמע המודרני כאשר עוסקים בהקלטות שמע או לחילופין בסינתזת צלילים קיים הצורך בחיווי של צורת האות וספקטרום האות אותו אנו מעבדים.

צורך זה נובע מהרצון להבנה מעמיקה של האות אתו אנו עובדים, וכחלק מתהליכי עיבוד האות כגון איזון (equalization) וויסות האנרגיה הספקטרלית, כיווץ התחום הדינאמי של האות, בזמן פעולת מיזוג בין אותות החופפים בספקטרום, וניטור פעולות של מסננים שונים המופעלים על האות.

בפרויקט זה ומתוך עיסוקי בעולם השמע ועיבוד האות בחרתי לממש נתח תדרים ואוסצילוסקופ עבור אותות שמע הנקלטים אל המחשב האישי דרך מיקרופון.

צורת הפתרון:

הפתרון המוצא מורכב משלוש יחידות בקרת אות:

1. אוסצילוסקופ (scope).
2. נתח תדרים (spectrum analyzer).
3. יחידת ניטור ממוצע האות בזמן (RMS).

אליהן מוזרמים הנתונים עבור כל חוצץ (buffer)דגימה שנקלט במערכת.

תדר הדגימה נבחר להיות [KHz]44.1 כמקובל בתחום השמע.

המשתמש בתכנית יכול לקבוע את עומק החוצץ בערכים הנעים החל מ $2^{10} = 1024$ דגימות ועד $2^{14} = 16384$ דגימות בקפיצות המהוות חזקות של 2 וזאת לצרכי חישוב מהירים של התמרת פורייה (למרות שההתמרה מחושבת בפונקציה מובנית של CVI ראיתי לנכון לעבוד עם חזקות שלמות של 2 מאחר וחישובי התמרות פורייה מהירים בהרבה בצורה זו).

הערכים נבחרו כך שניתן יהיה לקבל איזון בין רזולוציה בתדר ובין רזולוציה בזמן.

בתדר הדגימה הנבחר זמני הדגימה נעים בין [ms] 23 ובין [s] 0.37 ויוצא שרזולוציית התדר נעה בין [Hz]43 ובין [Hz]1.34 בהתאמה.

האמפליטודה מוצגת ביחידות של וולט המנורמלות לאחד כפי שמתקבל מהדגם.

ציר הזמן מוצג כאינדקסים של חלון הדגימה המוצג.

הספקטרום מוצג ביחידות של [dBFS] dB full scale כלומר "דציבל בסקאלה מלאה" כשנקודת הייחוס היא 0 דציבל עבור העוצמה המרבית האפשרית (1) כמקובל בעיבוד אותות דיגיטלי.

ציר התדר בגרף הספקטרום מוצג בין [KHz]20 - [Hz]20 כמקובל בתחום.

ניתנת האפשרות לשנות את הסקאלה של ציר התדר כסקאלה לינארית או לוגריתמית.

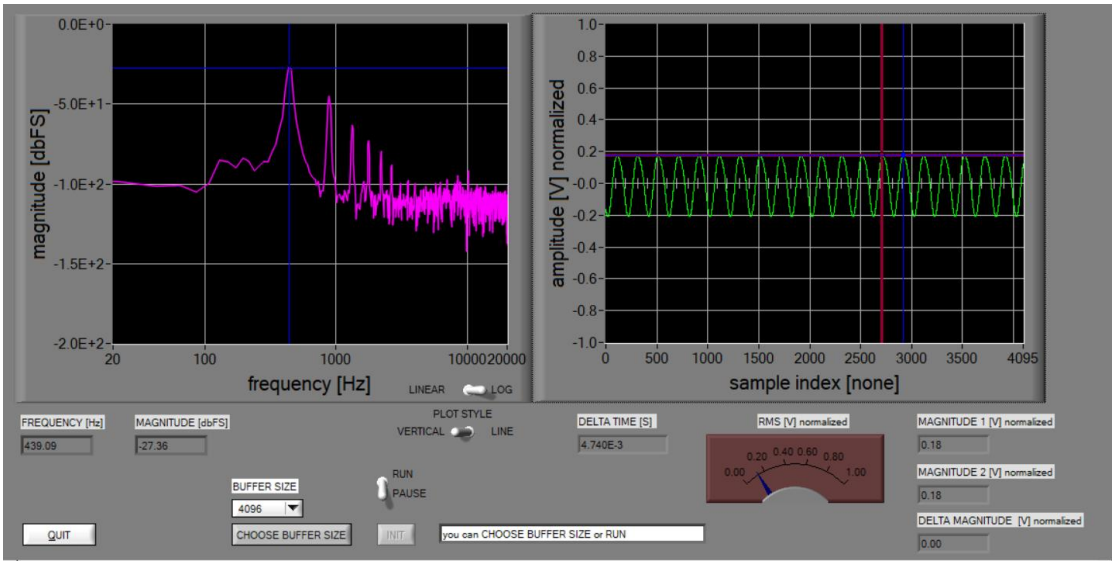
כאופציה נוספת למשתמש ניתנת בחירה לסוג הפלט בין גרף קווי ובין גרף עמודות.

המשתמש יכול לבצע חקירה של גרף האוסצילוסקופ במידה והוא לוחץ על הגרף מתקבלים נתוני הנקודות אותם בחר לסמן וכוללים אמפליטודות הפרש אמפליטודות והפרש זמנים בין שתי הסמנים.

המשתמש יכול לבצע חקירה של גרף נתח התדרים במידה והוא לוחץ על הגרף מתקבלת העוצמה והתדר בנקודה בה בחר.

בלחיצה על כפתור ctrl יחד עם בחירת מקטע בחלון המוצג אפשר המשתמש מקבל זום של אזור לצורך ניטור מדויק יותר.

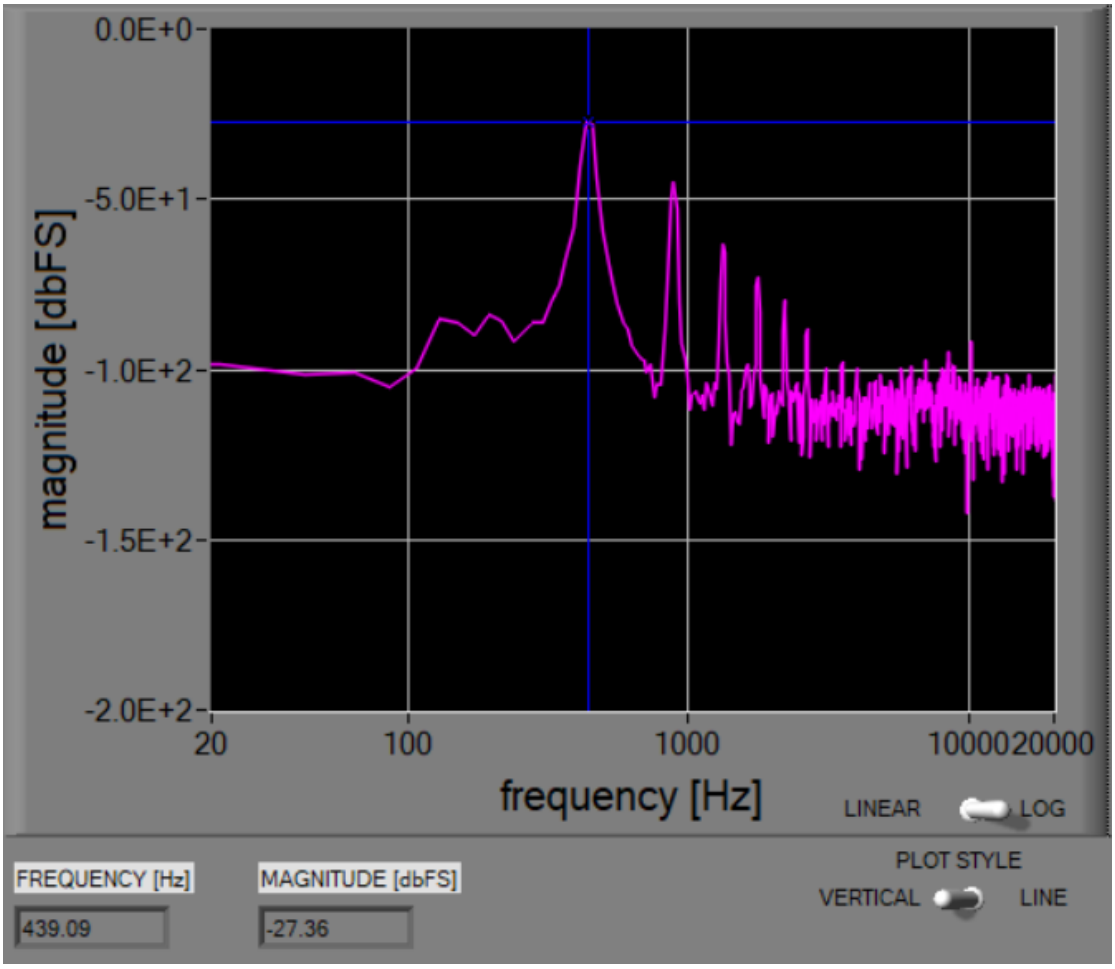
ממשק המשתמש:



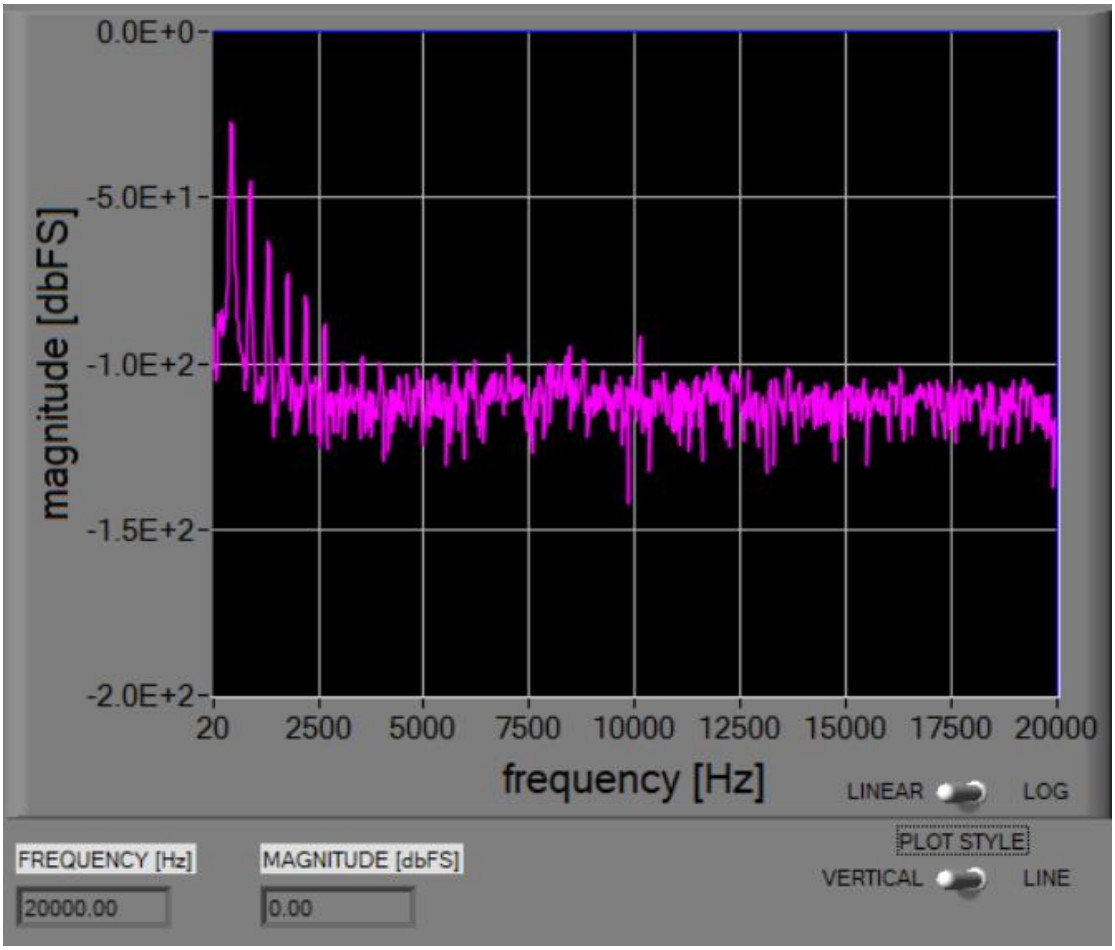
מציאת אמפליטודה ותדר ההרמוניה החזקה ביותר:

בדוגמא זו נקלט סינוס בעל תדר של כ [440[Hz ניתן לראות עוד מספר הרמוניות

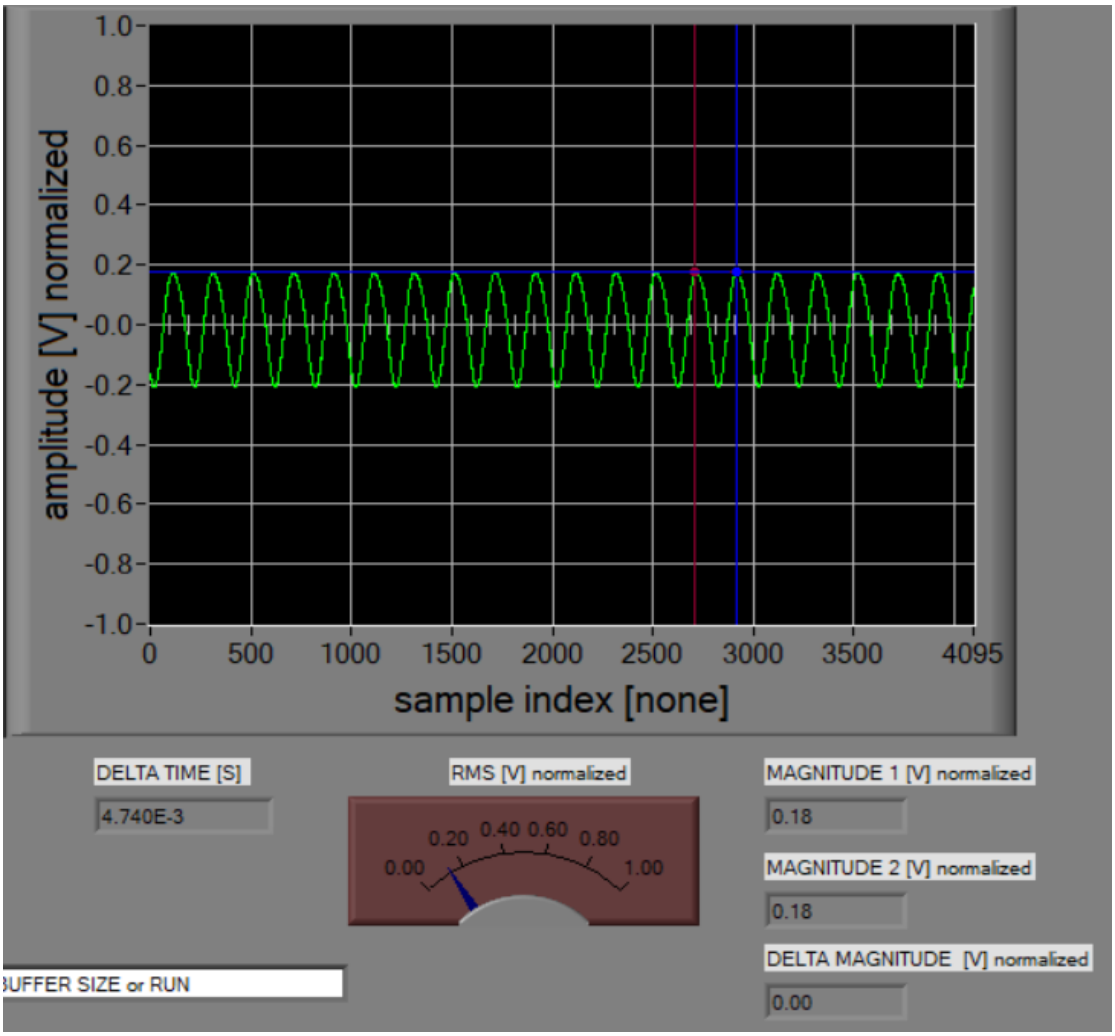
הנובעות מעיוות האות ע"י המיקרופון וכתוצאה מחלונות הדגימה.



בסקאלה לינארית:



מציאת אמפליטודה, הפרש אמפליטודות בין שתי נקודות, הפרש הזמנים בין הנקודות, ערך RMS של החלון:



חלקי התכנית:

1. חלק המטפל בממשק המשתמש, חלק זה מאפשר למשתמש אינטראקציה עם התוכנה, בה יכול המשתמש לבצע אתחול הגדרות, שימוש בכלים עבור ניטור האות וחקירת האות.
חלק זה נכתב בקפידה תוך שימת לב לפרטים והתחשבות ככל שהנני מודע למניעת באגים וקריסות.

2. תהליכון (thread) נפרד העוסק בעיבוד האותות הנקלטים ובשליחתם לממשק המשתמש, הליך זה היה נראה לי הדבר הנכון מתוך מאגר הכלים המוכר לי בעולם התכנות וכפי שנלמד בקורס, ואכן הממשק מתפקד ללא תקלות.
כאשר המשתמש לוחץ על RUN המערכת שולפת תהליכון מהבריכה ומקצה אותו עבור פונקציית התהליכון שנכתבה, בתהליכון זה האות הנדגם עובר חישובים בעזרת פונקציות הספרייה כאשר התהליך קורה בצורה הבאה, הכפלה חלון הדגימה בחלון המינג לשם הקטנת זליגה ספקטראלית, לאחר מכן מחושב ספקטרום האות ומתוכו מחושב הלוגריתם עבור כל תדר כך שהתוצאה מתקבלת ב dB.
כמו כן מחושב הממוצע בזמן ע"י שימוש בפונקציה המתאימה לכך.

3. חלקי אתחול וסגירת התכנית המהווים עניין חשוב באשר להקצאת המשאבים הנדרשים אתחולם כפי הנדרש וכמובן סגירתם באופן כזה שאינו יגרור זליגת זיכרון או קריסת התכנית.
פונקציית האתחול הנקראת כאשר המשתמש לוחץ INIT מקצה את המשאבים הנדרשים לתכנית בהתאם לבחירת המשתמש, בה מתקיים אתחול זרם הנתונים מהמיקרופון ומתבצעת פתיחתו במידה ולא התרחשו תקלות בתהליך.

פונקציית הסגירה נקראת בתגובה ללחיצת המשתמש על QUIT בתהליך זה מתבצעת סגירה ושחרור המשאבים שנדרשו לתכנית.

הבעיות ההנדסיות הכרוכות בפרויקט זה הינן:

1. הבנת תהליך קבלת הנתונים ממיקרופון המחשב והיכולת להתממשק עם זרם הנתונים המגיע ממיקרופון המחשב ולעשות בהם שימוש.
2. יצירת ממשק משתמש פרקטי אשר מתאים לניתוח אותות שמע ובעל סטנדרטים מהתחום.
3. שימוש בספריית קוד פתוח חיצונית ככלי עזר לפרויקט דבר אשר אני עושה לראשונה בפרויקט זה. החל מקומפילציית הספרייה וכלה בלימוד והבנת הספרייה והאפשרויות שהיא מקנה.
4. יכולת עבודה רציפה עם הממשק וזאת במקביל לזרימה השוטפת של הנתונים אל התוכנית.
5. סגירת התכנית ושחרור נכון של משאבי התכנית.

**פרויקט זה אינו מורכב מבחינה חישובית ומתמטית אך עוסק בפן הטכני של עבודה

בסביבת זמן אמת ממוחשבת אותו למדתי בזמן העבודה.

את הבעיות הנ"ל הצלחתי לפתור:

1. לאחר חיפושים ובירורים מצאתי את הספרייה PORTAUDIO שהינה ספריית קוד פתוח הנותנת מענה בכל הנוגע להתממשקות עם מאפייני השמע השונים במחשב האישי, ובכך שלמדתי לעבוד אתה קיבלתי את היכולת לביצוע המשימה של דגימת האותות מהמיקרופון.

2. כחלק מהקורס נלמדו נושאים להם הייתי זקוק לשם ביצוע הפרויקט ובהם עשיתי שימוש.
   א. שימוש בגרפים המובנים ב CVI ויישומם לצרכי הפרויקט שליטה בסקאלה ומתן יכולת למשתמש האם לצפות בסקאלה לינארית או לוגריתמית, כלי המקובל מאוד בתחום.
   ב. שימוש ב threads עבור הטיפול בזרימת הנתונים למערכת כך שהעבודה עם הממשק לא תפגע.
   ג. שליטה בממשק המשתמש למניעת באגים וקריסות כלומר לאפשר למשתמש לעבוד רק במסגרת אשר לא תגרור קריסת התכנית.
   ד. בעזרת ספריות העזר הכלולות במערכת ה NI CVI המאפשרות ניתוח ספקטרלי הנדרש לשם המוצא הרצוי.
   ה. בעזרת הכלים הידידותיים של CVI המטפלים בכל הקשור לעבודה מול ממשק המשתמש המאפשרים בניית ממשק בצורה ידידותית מהירה ונוחה.

3. לאחר קריאת מידע וחקירת נושא תהליך הקומפילציה עבור ספריות קוד פתוח ומספר ניסיונות לביצוע קומפילציה, הצלחתי לראשונה לעבור את תהליך הקומפילציה עבור הספרייה הנ"ל עבור המכונה עליה אני מתכנת ולאחר מכן לטעון אותה כחלק מסביבת הפיתוח לפרויקט.
   את התהליך הזה עד היום לא הצלחתי לבצע לבדי והוא נותן לי את האפשרות לבצע עבודות תכנות הנסמכות על ספריות חיצוניות כמקובל בעולם התוכנה.

   הבעיה העיקרית שנתקלתי בה הייתה הבנת דרך פעולת הספרייה PORTAUDIO ובפרט של תהליך הפסיקות וכיצד לכתוב את שגרת הקוד המופעל בתגובה לפסיקות המתקבלות מהדגם.
   לאחר קריאה באתר המפרסם את הספרייה, עיון בדוגמאות ובעזרת ניסוי ותהייה הגעתי למסקנות שהביאו לתוצאה הרצויה, ובסוף התהליך הבנתי את מהותו ורכשתי כלים המתאימים לביצוע עבודות עתידיות בעולם השמע.

4. בפרויקט קיים הצורך לייצר מערכת העובדת בקצב קבלת הנתונים מהדגם אשר אינה פוגעת ביכולת המשתמש להפעיל את התכנית, לשם כך עבדתי עם תהליכון נפרד המטפל בפליטת הנתונים למסך ועושה זאת בצורה מסונכרנת לגודל החוצץ וזמני הדגימה.

5. בחלק המטפל בסגירה נתקלתי בקושי מסוים כאשר המערכת הייתה קורסת בזמן יציאה מאחר וקיים הצורך לסגור את התהליכון בצורה אסינכרונית וזאת לפני שחרור הזיכרון כך שהתהליכון לא ייגש לזיכרון ששוחרר.
עובדה זו התבררה לי לאחר חקירת הקריסות וטופלה ע"י העלאת דגל כאשר המשתמש לוחץ על יציאה מהתכנית, דגל זה מזוהה בתוך התהליכון וגורם לסגירתו ובכך שיצרתי השהיה של כחצי שניה מזמן הלחיצה מובטח שהתהליכון יגיע לנקודה בא הוא מזהה את הצורך לסיום, השהיה זו אינה ארוכה כך שאינה מעכבת את הליך הסגירה בצורה משמעותית.
אינני יודע אם דרך זו מקובלת אך הביאה להפסקת הקריסות.

סיכום והצעות לשיפור:

לסיכום:

אוכל להגיד שאני מאוד מרוצה ושמח שבחרתי לבצע את הפרויקט הנ"ל ומרגיש שהתכנית נעשתה ברמה מקצועית ובסטנדרטים מהתחום.

קיבלתי כלים תכנותיים פרקטיים הקשורים לתחום העניין שלי בהנדסת קול ושמע

וההתמודדות עם פרויקט זה סיפקה לי כלים בתחום זה אשר יעזרו לי לשם ביצוע פרויקט הגמר לתואר העוסק בעיבוד אותות שמע וגם לפרויקטים עתידיים שיש בכוונתי לבצעם.

הצעות לשיפור:

התכנית במצבה הנוכחי מתחברת למיקרופון ברירת המחדל שמוגדר במערכת.

הייתי מכניס לפרויקט זה את מתן האפשרות למשתמש לבחור את מקור השמע הנכנס למערכת כלומר האם לדגום נתונים מהמיקרופון הרגיל, מכרטיס קול חיצוני או מתוך זרם נתונים פנימי המופק בתכנית מחשב אחרת.

רשימת הפונקציות מסביבת CVI המהותיות לתכנית:

Controls graphs strip charts:

SetCtrlAttribute, SetCtrlVal, GetCtrlVal, GetGraphCursor, DeleteGraphPlot, PlotY, PlotXY.

Advanced analysis library:

שימוש לצורך הפחתת השפעת קצוות חלון הדגימה על התמרת פורייה .- HanWin

פונקציה המחשבת את ספקטרום האות בצורה יעילה. - Spectrum

מחשבת את ממוצע האות בזמן. - RMS

Utility library:

מבצעת השהיה לפי הערך הנדרש. – Deley

מודיעה לבריכת התהליכונים על הפעלת פונקציה בתהליכון מתוך הבריכה. – CmtScheduleThreadPoolFunction

CmtWaitForThreadPoolFunctionCompletion

מחכה לסיום התהליכון כך שלאחריה אפשר יהיה לסגור אותו בבטחה ולשחרר את תעודת הזהות שהוקצתה לו.

CmtReleaseThreadPoolFunctionID

משחררת את תעודת הזהות שהוקצתה לתהליכון ומחזירה אותה לבריכה.

CmtExitThreadPoolThread

סוגרת את התהליכון.

קובץ ההגדרות defines.h :

```
 #include <stdio.h>
#include <stdlib.h>
#include "portaudio.h"

#define SAMPLE_RATE  (44100)
#define NUM_SECONDS     (10)
#define NUM_CHANNELS    (1)
/* #define DITHER_FLAG     (paDitherOff) */
#define DITHER_FLAG     (0) /**/

/* Select sample format. */
#if 1
#define PA_SAMPLE_TYPE  paFloat32
typedef float SAMPLE;                    ///was float
#define SAMPLE_SILENCE  (0.0f)
#define PRINTF_S_FORMAT "%.8f"
#elif 1
#define PA_SAMPLE_TYPE  paInt16
typedef short SAMPLE;
#define SAMPLE_SILENCE  (0)
#define PRINTF_S_FORMAT "%d"
#elif 0
#define PA_SAMPLE_TYPE  paInt8
typedef char SAMPLE;
#define SAMPLE_SILENCE  (0)
#define PRINTF_S_FORMAT "%d"
#else
#define PA_SAMPLE_TYPE  paUInt8
typedef unsigned char SAMPLE;
#define SAMPLE_SILENCE  (128)
#define PRINTF_S_FORMAT "%d"
```

```
#endif

typedef struct
{
   int        frameIndex;  /* Index into sample array. */
   int        maxFrameIndex;
   SAMPLE     *recordedSamples;
}
paTestData;
////////////////////////////////////////////////////////////////// my parameters
#define low_limit 0.00000000000000000001
static int                                          panelHandle;
static int                                          start1;
static float                                *x_axis ;

static int                                  init_flag;
static int                                  plot_style;
static double                               *fft_re,*time_series;
static int                                          my_err;
////////////////////////////////////////////////////////////////// portaudio parameters

static  PaStreamParameters  *inputParameters ;//*outputParameters //used whan output sound
static PaStream*          stream;
static PaError            err = paNoError;
static paTestData         data;
static int              i;
static int              totalFrames;
static int              numSamples;
static int              numBytes;
static int                                  SAMPLES_PER_BUFFER=1024;

////////////////////////////////////////////////////////////////// my functions
int CVICALLBACK ThredFunction (void *functionData);

void close_stream(PaStream* my_stream) ;
```

```
static int audio_in_Callback( const void *inputBuffer,
                                                void *outputBuffer,
                unsigned long framesPerBuffer,
                const PaStreamCallbackTimeInfo *timeInfo,
                PaStreamCallbackFlags statusFlags,
                void *userData );
```

קובץ התכנית io.c :

```
#include <utility.h>
#include <analysis.h>
#include <ansi_c.h>
#include <cvirte.h>
#include <userint.h>
#include "io1.h"

#include "defines.h"

//////////////////////////////////////////////////////////////////////
int main (int argc, char *argv[])
{
        if (InitCVIRTE (0, argv, 0) == 0)
                return -1;          /* out of memory */
        if ((panelHandle = LoadPanel (0, "io1.uir", PANEL)) < 0)
                return -1;

        DisplayPanel (panelHandle);
        RunUserInterface ();
        DiscardPanel (panelHandle);

        return 0;
}
//////////////////////////////////////////////////////////////////////////////////
int CVICALLBACK QuitCallback (int panel, int control, int event,
```

```
                                                                    void *callbackData, int eventData1, int
eventData2)
{
        switch (event)
        {
                case EVENT_COMMIT:
                        quit=1;
                        Delay(0.5);                     //make shure thred if finished and its safe to
free memory
                        close_stream(stream);
                        QuitUserInterface (0);

                        break;
                case EVENT_RIGHT_CLICK:

                        break;
        }
        return 0;
}
////////////////////////////////////////////////////////////////////////////////
int CVICALLBACK INIT_callback (int panel, int control, int event,
                                                                    void *callbackData, int eventData1, int
eventData2)
{
        int k;

        switch (event)
        {
                case EVENT_COMMIT:
                        init_flag=1;
                        x_axis = (float*)malloc(sizeof(float)*SAMPLES_PER_BUFFER/2);
                        time_series =(double*)malloc(sizeof(double)*SAMPLES_PER_BUFFER);
                        fft_re = (double*)malloc(sizeof(double)*SAMPLES_PER_BUFFER);

                        if(x_axis != NULL && time_series != NULL && fft_re != NULL)
                        {
```

```
                    for( k=0;k < SAMPLES_PER_BUFFER/2 ;k++)
                    {
                        x_axis[k] =
2*(float)k*((float)SAMPLE_RATE/(float)SAMPLES_PER_BUFFER);
                        //frequency axis calculation (cast to float in order to get
correct ans)
                    }
                }
                else
                {
                  my_err=-10;
                    goto done;
                }
                data.maxFrameIndex = totalFrames = SAMPLES_PER_BUFFER;
            data.frameIndex = 0;
            numSamples = totalFrames * NUM_CHANNELS;
            numBytes = numSamples * sizeof(SAMPLE);


                data.recordedSamples = (SAMPLE *) malloc( numBytes ); /* From now on,
recordedSamples is initialised. */
                if( data.recordedSamples == NULL )
                {
                    my_err=-10;
                    goto done;
                }
                else
                {
                    for( i=0; i<numSamples; i++ )
                    data.recordedSamples[i] = 0;
                }


                err = Pa_Initialize();
                if( err != paNoError )
                    goto done;
```

```
inputParameters =
(PaStreamParameters*)malloc(sizeof(PaStreamParameters));

inputParameters->device = Pa_GetDefaultInputDevice(); /* default input
device */

if (inputParameters->device == paNoDevice)

        goto done;


inputParameters->channelCount = 2;          /* as if we record stereo
input */

inputParameters->sampleFormat = PA_SAMPLE_TYPE;

inputParameters->suggestedLatency = Pa_GetDeviceInfo(
inputParameters->device )->defaultLowInputLatency;

inputParameters->hostApiSpecificStreamInfo = NULL;


/* Record audio. */


err = Pa_OpenStream(

                &stream,

                inputParameters,

                NULL,SAMPLE_RATE,

            SAMPLES_PER_BUFFER,

            paClipOff,     /* we won't output out of range samples so don't
bother clipping them */

            audio_in_Callback,

            &data );

if( err != paNoError )

        goto done;


err = Pa_StartStream( stream );

if( err != paNoError )

        goto done;


SetCtrlAttribute (panelHandle, PANEL_BINARYSWITCH, ATTR_DIMMED, 0);

SetCtrlAttribute (panelHandle, PANEL_init, ATTR_DIMMED, 1);

SetCtrlVal (panelHandle, PANEL_STRING,"please press the RUN button" );


done:
```

```
                        if(err<0)
                        close_stream(stream);
                        break;
        }
        return 0;
}
////////////////////////////////////////////////////////////////////////////////
int CVICALLBACK START_STOP_button (int panel, int control, int event,
                                                        void *callbackData, int eventData1,
int eventData2)
{

        switch (event)
        {
                case EVENT_COMMIT:

                        start1 = !start1;

                        if(start1)
                        {
                                SetCtrlVal (panelHandle, PANEL_STRING,"you can PAUSE the plot"
);
                                CmtScheduleThreadPoolFunction
(DEFAULT_THREAD_POOL_HANDLE, ThredFunction, NULL, &tf_handler);

                                SetCtrlAttribute (panelHandle, PANEL_init, ATTR_DIMMED, 1);
                                SetCtrlAttribute (panelHandle, PANEL_buffer, ATTR_DIMMED, 1);
                                SetCtrlAttribute (panelHandle, PANEL_ch_buff_btn,
ATTR_DIMMED, 1);
                        }
                        else
                        {
                                CmtWaitForThreadPoolFunctionCompletion
(DEFAULT_THREAD_POOL_HANDLE,tf_handler, OPT_TP_PROCESS_EVENTS_WHILE_WAITING);
                                CmtReleaseThreadPoolFunctionID
(DEFAULT_THREAD_POOL_HANDLE, tf_handler);
                                SetCtrlAttribute (panelHandle, PANEL_buffer, ATTR_DIMMED, 0);
```

```
                                        SetCtrlAttribute (panelHandle, PANEL_ch_buff_btn,
ATTR_DIMMED, 0);
                                        SetCtrlVal (panelHandle, PANEL_STRING,"you can CHOOSE
BUFFER SIZE or RUN" );


                                }
                                break;
                        case EVENT_RIGHT_CLICK:


                                break;
                }
                return 0;
}
//////////////////////////////////////////////////////////////////////////////////
int CVICALLBACK graph_callback (int panel, int control, int event,
                                                                        void *callbackData, int
eventData1, int eventData2)
{   static double X,Y;
                switch (event)
                {
                        case EVENT_COMMIT:
                                GetGraphCursor (panelHandle, PANEL_GRAPH, 1,&X ,&Y );
                                SetCtrlVal (panelHandle, PANEL_mag, Y);
                                SetCtrlVal (panelHandle, PANEL_freq, X);


                                break;
                        case EVENT_RIGHT_CLICK:


                                break;
                }
                return 0;
}
//////////////////////////////////////////////////////////////////////////////////
int CVICALLBACK graph_2_callback (int panel, int control, int event,
                                                                        void *callbackData, int
eventData1, int eventData2)
```

```
{   static double X1,Y1,X2,Y2;
        switch (event)
        {
                case EVENT_COMMIT:
                        GetGraphCursor (panelHandle, PANEL_GRAPH_2, 1, &X1, &Y1);
                        GetGraphCursor (panelHandle, PANEL_GRAPH_2, 2, &X2, &Y2);
                        SetCtrlVal (panelHandle, PANEL_curr1, Y1);
                        SetCtrlVal (panelHandle, PANEL_curr2, Y2);
                        Y2=Y1-Y2;
                        X2=(X1-X2)/SAMPLE_RATE;
                        SetCtrlVal (panelHandle, PANEL_delta_v, Y2);
                        SetCtrlVal (panelHandle, PANEL_delta_t, X2);
                        break;
                case EVENT_RIGHT_CLICK:

                        break;
        }
        return 0;
}
//////////////////////////////////////////////////////////////////////////////////////////
int CVICALLBACK ThredFunction (void *functionData)



{
        int j;
        static double rms_val;

                while(start1)
                {
                        if(quit)
                                CmtExitThreadPoolThread (0);

                        if ( ( err = Pa_IsStreamActive( stream ) ) == 1)
                        {
                                for(j=0;j<SAMPLES_PER_BUFFER;j++)
```

```
{
        fft_re[j] = (double)data.recordedSamples[j];
}

memcpy ( time_series, fft_re,
sizeof(double)*SAMPLES_PER_BUFFER);

HanWin (fft_re, SAMPLES_PER_BUFFER);//pre processing to avoid
the rect window effects
Spectrum (fft_re,SAMPLES_PER_BUFFER ); //magnutude spectrum
calculation

RMS (time_series,SAMPLES_PER_BUFFER ,&rms_val );
SetCtrlVal (panelHandle, PANEL_NUMERICMETER, rms_val);

DeleteGraphPlot (panelHandle, PANEL_GRAPH, -1,
VAL_DELAYED_DRAW);
DeleteGraphPlot (panelHandle, PANEL_GRAPH_2, -1,
VAL_DELAYED_DRAW);
for(j=0;j<SAMPLES_PER_BUFFER;j++)
{
        if(fft_re[j]<low_limit)
        {
                fft_re[j]=-200;//floor
        }
else
        {
                fft_re[j] = 10*log10 (fft_re[j]);
        }
}

PlotY (panelHandle, PANEL_GRAPH_2, time_series,
SAMPLES_PER_BUFFER, VAL_DOUBLE, VAL_THIN_LINE, VAL_EMPTY_SQUARE, VAL_SOLID, 1,
VAL_GREEN);
if(plot_style)
{
```

```
                                        PlotXY (panelHandle, PANEL_GRAPH, x_axis, fft_re,
SAMPLES_PER_BUFFER/2, VAL_FLOAT, VAL_DOUBLE, VAL_VERTICAL_BAR, VAL_EMPTY_SQUARE,
VAL_SOLID, 1,

        VAL_MAGENTA);
                                }
                                else
                                {
                                        PlotXY (panelHandle, PANEL_GRAPH, x_axis, fft_re,
SAMPLES_PER_BUFFER/2, VAL_FLOAT, VAL_DOUBLE, VAL_THIN_LINE, VAL_EMPTY_SQUARE,
VAL_SOLID, 1, VAL_MAGENTA);
                                }



                                data.frameIndex = 0;


                        }


                        Delay(SAMPLES_PER_BUFFER/SAMPLE_RATE);


                }
        if(quit)
                CmtExitThreadPoolThread (0);


        return 0;
}
////////////////////////////////////////////////////////////////////////////////
int CVICALLBACK change_buffer_size_callback (int panel, int control, int event,
                                                                        void *callbackData, int
eventData1, int eventData2)
{


        switch (event)
        {
                case EVENT_COMMIT:
                        GetCtrlVal (panelHandle, PANEL_buffer, &SAMPLES_PER_BUFFER);
                        if(init_flag == 1)
```

```
                        {
                                close_stream(stream);
                        }
                        SetCtrlAttribute (panelHandle, PANEL_init, ATTR_DIMMED, 0);
                        SetCtrlAttribute (panelHandle, PANEL_BINARYSWITCH, ATTR_DIMMED, 1);
                        SetCtrlVal (panelHandle, PANEL_STRING,"please press the INIT button" );
                        break;
                case EVENT_RIGHT_CLICK:

                        break;
        }
        return 0;
}
////////////////////////////////////////////////////////////////////////////////
int CVICALLBACK plot_style_callback (int panel, int control, int event,
                                                                        void *callbackData,
int eventData1, int eventData2)
{
        switch (event)
        {
                case EVENT_COMMIT:
                        plot_style = !plot_style;
                        break;
                case EVENT_RIGHT_CLICK:

                        break;
        }
        return 0;
}
////////////////////////////////////////////////////////////////////////////////
int CVICALLBACK x_log_callback (int panel, int control, int event,
                                                                void *callbackData, int
eventData1, int eventData2)
{   int val;
        switch (event)
```

```
	{
		case EVENT_COMMIT:
			GetCtrlVal (panelHandle, PANEL_x_log, &val);
			if(val)
			{
				SetCtrlAttribute (panelHandle,
PANEL_GRAPH,ATTR_XMAP_MODE, VAL_LOG);
			}
			else
			{
				SetCtrlAttribute (panelHandle, PANEL_GRAPH,
ATTR_XMAP_MODE, VAL_LINEAR);
			}
			break;
		case EVENT_RIGHT_CLICK:


			break;
	}
	return 0;
}
////////////////////////////////////////////////////////////////////////////////
int CVICALLBACK buffer_size_val (int panel, int control, int event,
								void *callbackData, int
eventData1, int eventData2)
{
	switch (event)
	{
		case EVENT_COMMIT:
			SetCtrlVal (panelHandle, PANEL_STRING,"please press the CHOOSE BUFFER
SIZE button" );
			break;
		case EVENT_RIGHT_CLICK:


			break;
	}
	return 0;
```

```
}

////////////////////////////////////////////////////////////////////////////////
void close_stream(PaStream* my_stream)
{

		if(fft_re!=NULL)
		{
			free(fft_re);
			fft_re=NULL;
		}

		if(time_series!=NULL)
		{
			free(time_series);
			time_series=NULL;
		}

		if(x_axis!=NULL)
		{
			free(x_axis);
			x_axis=NULL;
		}

		if( data.recordedSamples!=NULL )
		{	/* Sure it is valid pointer. */
			free( data.recordedSamples );
			data.recordedSamples = NULL;
		}

		if( err != paNoError )
		{
			SetCtrlVal (panelHandle, PANEL_STRING,"An error occured while using the
portaudio stream" );
		}
```

```
                    if(my_err == -10)
                    {
                            SetCtrlVal (panelHandle, PANEL_STRING,"error allocating memory" );
                    }

                    err = Pa_CloseStream( my_stream );
                    Pa_Terminate();
}
////////////////////////////////////////////////////////////////////////////////
static int audio_in_Callback( const void *inputBuffer,
                                                        void *outputBuffer,
                unsigned long framesPerBuffer,
                const PaStreamCallbackTimeInfo *timeInfo,
                PaStreamCallbackFlags statusFlags,
                void *userData )
{
   paTestData *data = (paTestData*)userData;
   const SAMPLE *read_ptr = (const SAMPLE*)inputBuffer;
   SAMPLE *write_ptr = &data->recordedSamples[data->frameIndex * NUM_CHANNELS];
          //(void) outputBuffer; /* Prevent unused variable warnings. */
   //(void) timeInfo;
   //(void) statusFlags;
   //(void) userData;
          //long i;

   if( inputBuffer == NULL ) //in case of mised buffer
   {
      for( i=0; i<data->maxFrameIndex; i++ )
      {
         *write_ptr++ = SAMPLE_SILENCE;

      }
   }
   else
```

```
    {
                    memcpy(write_ptr , read_ptr , sizeof(SAMPLE)*framesPerBuffer);
    }

        return paContinue;

}
```

קובץ io1.h

```
/*************************************************************************/
 */LabWindows/CVI User Interface Resource (UIR) Include File/*
/*                                                                       */
 */WARNING: Do not add to, delete from, or otherwise modify the contents/*
        */of this include file/*                                        .
/*************************************************************************/

#include <userint.h<

#ifdef __cplusplus
    extern "C} "
#endif

 */     Panels and Controls/* :

#define  PANEL                    1
#define  PANEL_init               2      /* control type: command, callback function: INIT_callback/*
#define  PANEL_GRAPH              3      /* control type: graph, callback function: graph_callback/*
#define  PANEL_QUITBUTTON_2       4      /* control type: command, callback function:
QuitCallback/*
#define  PANEL_BINARYSWITCH       5      /* control type: binary, callback function:
START_STOP_button/*
#define  PANEL_STRING             6      /* control type: string, callback function: (none)/*
#define  PANEL_delta_t            7      /* control type: numeric, callback function: (none)/*
#define  PANEL_delta_v            8      /* control type: numeric, callback function: (none)/*
```

```
#define  PANEL_curr2                9      /* control type: numeric, callback function: (none)/*

#define  PANEL_curr1                10     /* control type: numeric, callback function: (none)/*

#define  PANEL_mag                  11     /* control type: numeric, callback function: (none)/*

#define  PANEL_freq                 12     /* control type: numeric, callback function: (none)/*

#define  PANEL_GRAPH_2              13     /* control type: graph, callback function:
graph_2_callback/*

#define  PANEL_ch_buff_btn          14     /* control type: command, callback function:
change_buffer_size_callback/*

#define  PANEL_x_log                15     /* control type: binary, callback function: x_log_callback/*

#define  PANEL_buffer               16     /* control type: ring, callback function: buffer_size_val/*

#define  PANEL_NUMERICMETER         17     /* control type: scale, callback function: (none)/*

#define  PANEL_plot_style           18     /* control type: binary, callback function:
plot_style_callback/*



 */     Control Arrays/* :


) */          no control arrays in the resource file/* (



 */     Menu Bars, Menus, and Menu Items/* :


) */          no menu bars in the resource file/* (



 */     Callback Prototypes/* :


int  CVICALLBACK buffer_size_val(int panel, int control, int event, void *callbackData, int eventData1,
int eventData2);

int  CVICALLBACK change_buffer_size_callback(int panel, int control, int event, void *callbackData, int
eventData1, int eventData2);

int  CVICALLBACK graph_2_callback(int panel, int control, int event, void *callbackData, int
eventData1, int eventData2);

int  CVICALLBACK graph_callback(int panel, int control, int event, void *callbackData, int eventData1,
int eventData2);

int  CVICALLBACK INIT_callback(int panel, int control, int event, void *callbackData, int eventData1, int
eventData2);
```

```
int  CVICALLBACK plot_style_callback(int panel, int control, int event, void *callbackData, int
eventData1, int eventData2);

int  CVICALLBACK QuitCallback(int panel, int control, int event, void *callbackData, int eventData1, int
eventData2);

int  CVICALLBACK START_STOP_button(int panel, int control, int event, void *callbackData, int
eventData1, int eventData2);

int  CVICALLBACK x_log_callback(int panel, int control, int event, void *callbackData, int eventData1,
int eventData2);



#ifdef __cplusplus

{

#endif
```