המכללה האקדמית להנדסה בראודה בכרמיאל

המחלקה להנדסת חשמל ואלקטרוניקה

תכנות מעבדי DSP

P12 – Heartbeat vs Breath Sound

פרנסיס עבוד

בשארה חביב

מרצה: יצחק קרוין

תאריך: 19/01/2025

תוכן עניינים

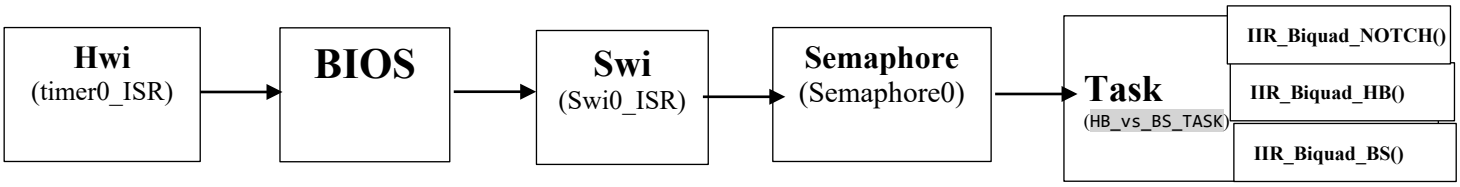
1. דרישות הפרויקט 3
2. דיאגרמת בלוקים של התוכנה 3
3. תרשים זרימה של המערכת 4
4. שיקולי תכנון 6
5. שיקולי התכנון עבור המסננים 6
6. שיקולי התכנון עבור התוכנית 10
7. תצורות 10
8. פונקציות התוכנית 13
9. תוצאות ההרצה 18
10. Execution graph, Task and CPU load 20
11. Execution graph 20
12. Task load 21
13. CPU load 21

דרישות הפרויקט

- ✓ השמת אות של פעימות הלב ו- נשימה נתון במערך בגודל 4405 דגימתו בתדר דגימה של 2000Hz
- ✓ שימוש ב- Timer אשר דוגם את האות בתדר הדגימה שנקבע ובאופן רציף וציקלי
- ✓ שימוש במסננים מתאימים ובכלי DSP אחרים כדי לזהות את פעימות הלב והנשימה בנפרד
- ✓ התוצאות צריכות להיות ארבעה גרפים. שניים הם האותות לאחר ההפרדה והשניים האחרים הם האנרגיה של האותות הפיזיולוגיים
- ✓ הצג גם, גרף FFT Magnitude של הקלט ולאחר סינון
- ✓ הצגת Execution graph, CPU load, Task load

דיאגרמת בלוקים של התוכנה

דיאגרמת הבלוקים בדגש על ארכיטקטורת multi-threading בזמן אמת:



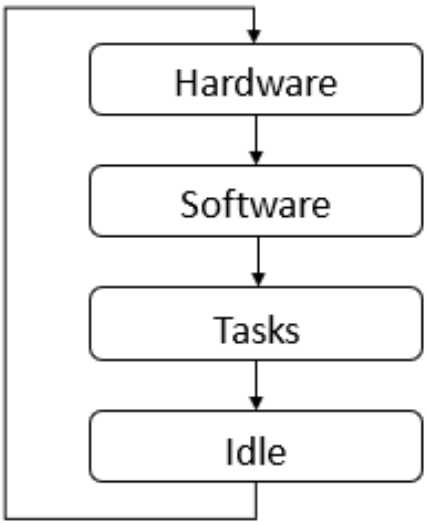
איור מס. 1 דיאגרמת הבלוקים של התוכנה

תרשים זרימה של המערכת

ה- Priority של ה- threads במערכת RTOS הינה בסדר הבא:

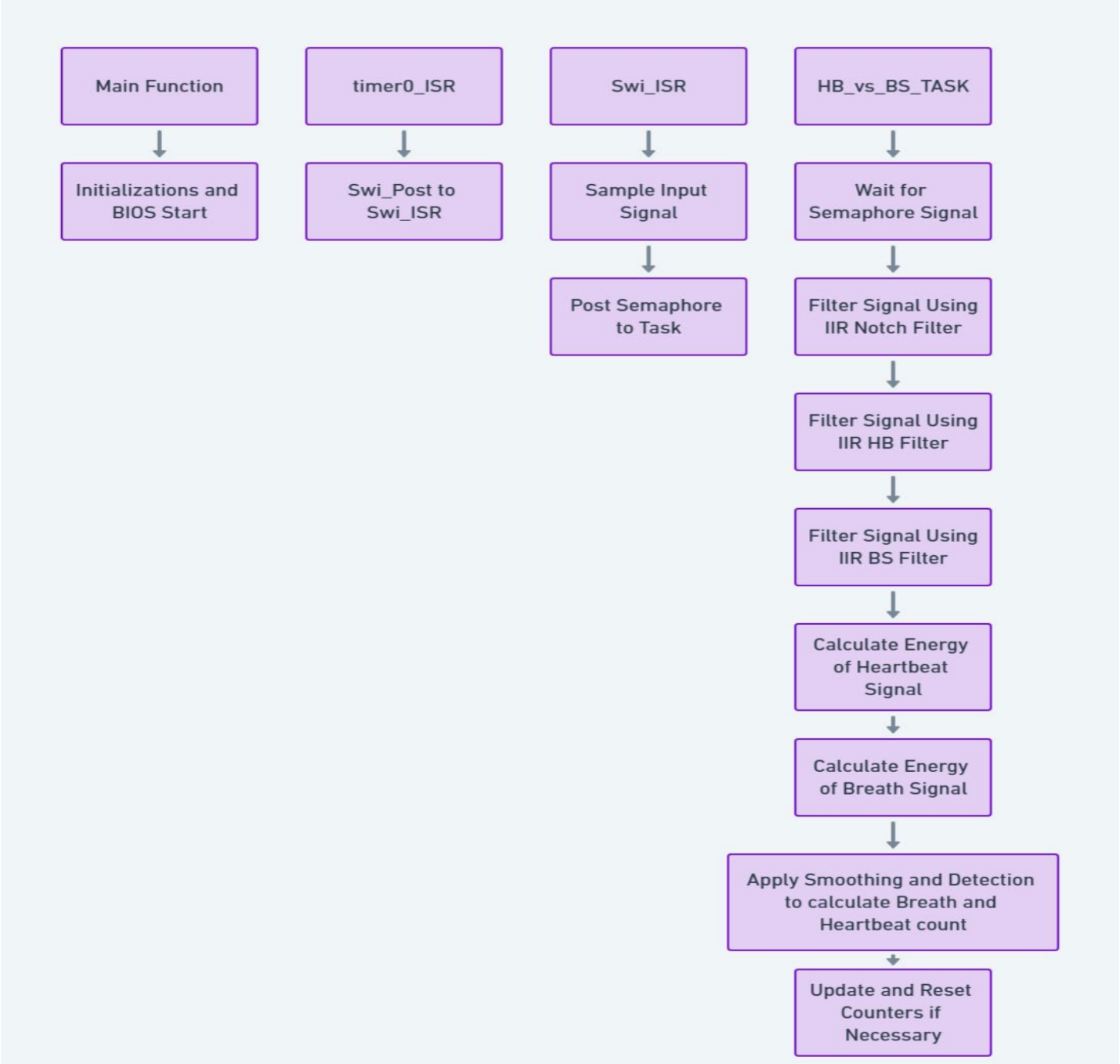
איור מס. 2 ה- priority של ה-threads

וסדר ביצוע הפעולות הינו:



איור מס. 3 סדר ביצוע ה-threads בכל הרצה

תרשים הזרימה של התוכנית:



איור מס. 4 תרשים הזרימה של התוכנית

שיקולי תכנון

מטרת הפרויקט היא להשתמש ב מערך של פעימות הלב ו- נשימה לסנן אותו ולהפריד בין פעימות הלב והנשימה ו- לזהות אותם בנפרד ו להוציא גרפים של האותות והאנרגיה שלהם, ביצענו את תכנון המסנן באמצעות אפליקציית Filter Designer ב- MATLAB.

מסנן ראשון הוא מסנן מסוג Single Notch IIR בתדר Fnotch=0.25KHz עם FS=2KHz ואת סדר המסנן מוגדר להיות 2 עם Q=1 לסנן את ההפרעה שהיתה בתדר בודד.

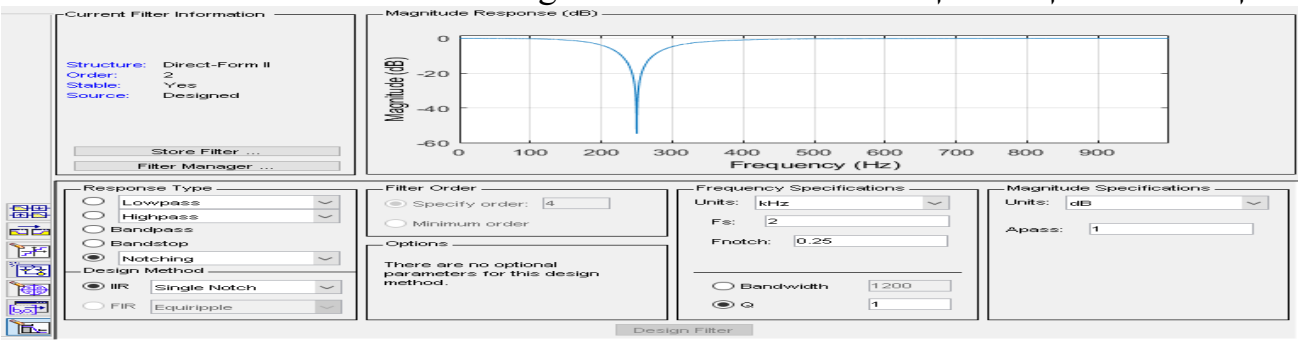
מסנן שני הוא מסנן מסוג IIR אליפטי LPF בתדר 135Hz ואת סדר המסנן בחרנו להיות 4 להפריד את אות של פעימות הלב מהאות המקורי.

מסנן שלישי הוא מסנן מסוג IIR אליפטי HPF בתדר 135Hz ואת סדר המסנן בחרנו להיות 4 להפריד את אות של הנשימה מהאות המקורי.

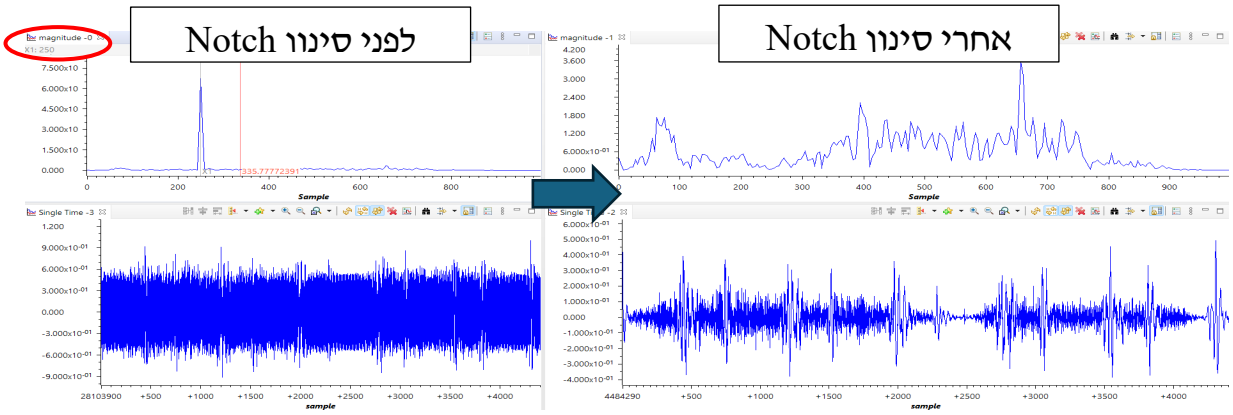
- בחרנו תדר 135 Hz למרות שהיה נתון בהגדרת הפרויקט 140 Hz מכיוון שסיננו את ההפרעה הנוטש פילטר ולכן הגבול היה 135 Hz.

שיקולי התכנון עבור המסננים

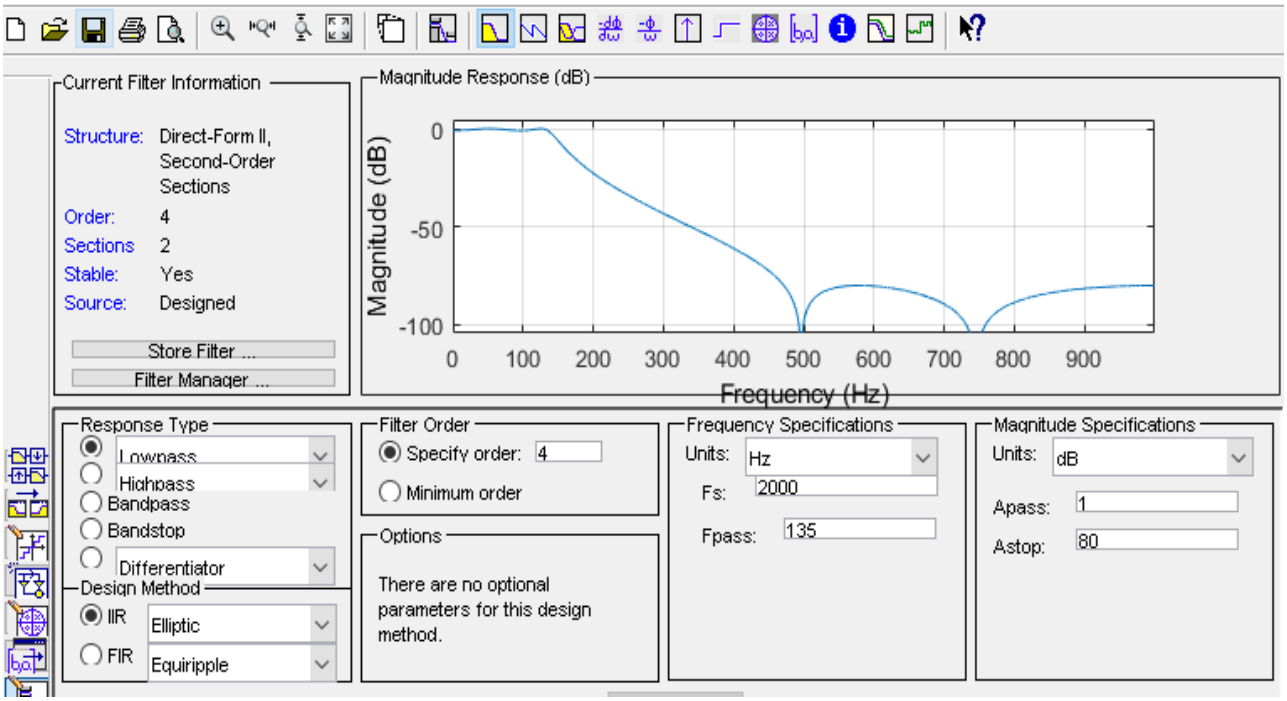
להלן תצוגת המסנן הראשון הנבחר כפי שמופיעה ב- Filter Designer ב-MATLAB:



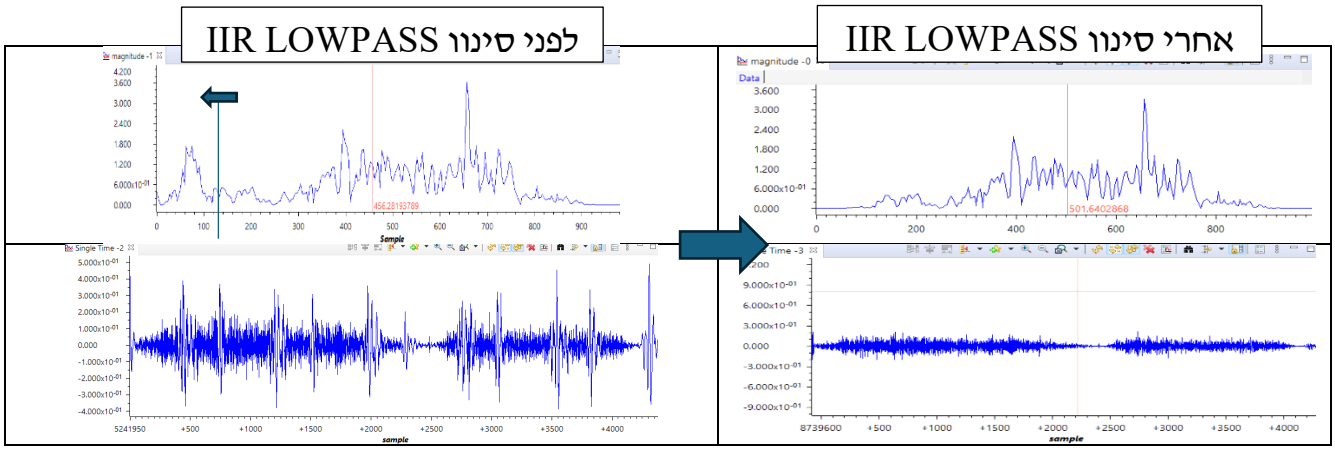
קבענו את ה Fnotch לפי ה Graph → FFT Magnitude אנחנו יכולים לראות את ההפרעה ב X1 = 250 Sample → Fnotch = 0.25KH



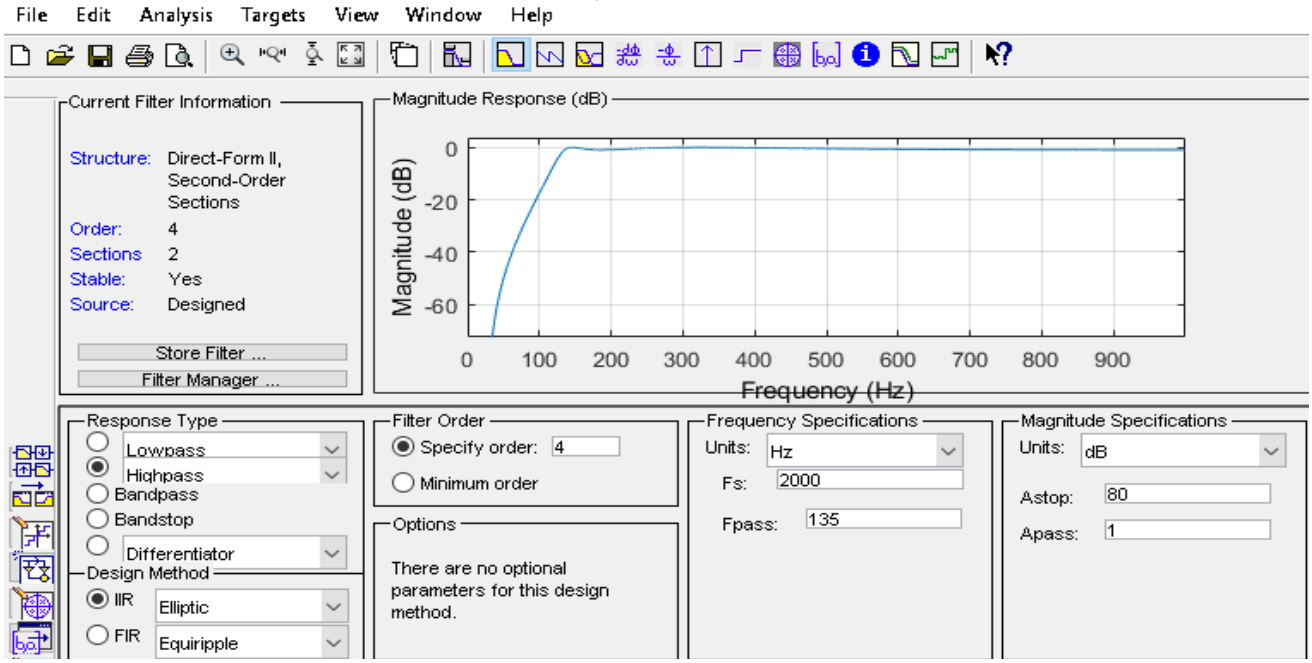
להלן תצוגת המסנן השני הנבחר כפי שמופיעה ב- Filter Designer ב-MATLAB:



בחרנו את התדר שיהיא Fpass = 135Hz כדי לקבל רק אות של הנשימה

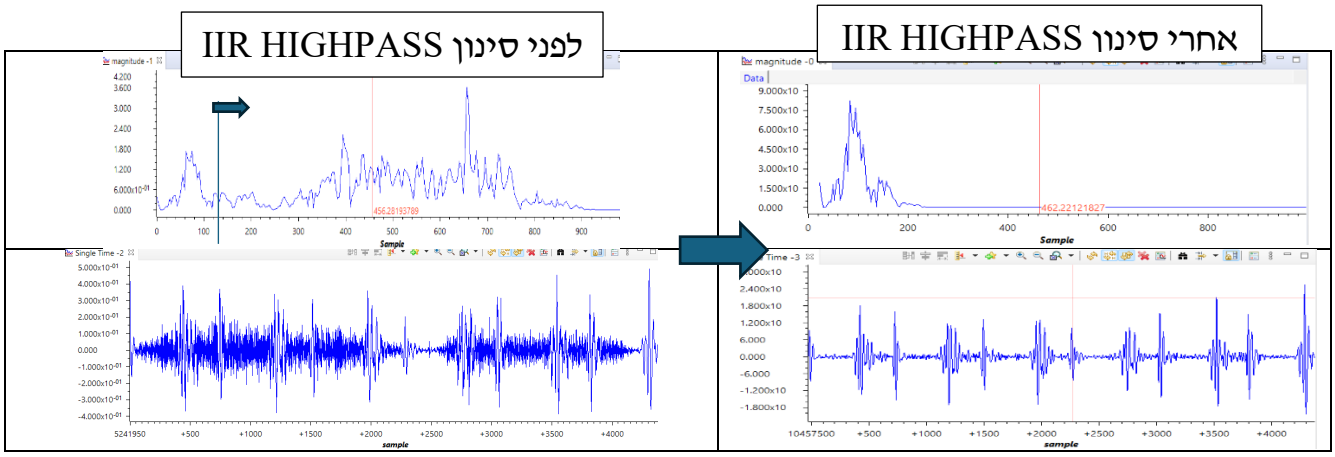


להלן תצוגת המסנן השני הנבחר כפי שמופיעה ב- Filter Designer ב-MATLAB :



איור מס. 5 תכנון המסנן ב- Filter Designer ב- MATLAB

בחרנו את התדר שיהיא Fpass = 135Hz ב IIR HIGHPASS כדי לקבל רק אות של פעימות הלב



תכנון המסנן בוצע במספר שלבים:

1. **הוספת מסנן Notch בתחילת השרשרת**

ראשית, בחרנו למקם מסנן Notch (מסנן דיכוי תדר ספציפי) כדי לסנן רעשי רשת או תדר לא רצוי אחר המופיע באות. מסנן זה מוציא את הרכיב המזיק לפני המשך הסינון וההפרדה של אות הנשימה ואות פעימות הלב.

2. **ניתוח תדרי האותות**

לאחר מכן הסתכלנו על האות במישור התדר, וגם נעזרנו במידע מהאינטרנט ומהמרצה, והבנו כי:

תדרי אות פעימות הלב (Heartbeat) קטנים מ־135Hz

תדרי אות הנשימה (Breath) גדולים מ־135Hz

3. **החלטה על סדר המסנן**

בחרנו שהסדר (Order) של המסננים יהיה 4 כדי לקבל רמת סינון מספקת והנחתה טובה באזורים הלא רצויים.

4. **בחירת סוג המסנן (IIR)**

העדפנו להשתמש במסנני IIR מכיוון שהם יעילים יותר מבחינה חישובית ודורשים מספר קטן יחסית של מקדמים (Coefficients) להשגת ביצועים טובים.

5. **בחירת מסנן אליפטי (Elliptic)**

מבין משפחות המסננים האפשריות, החלטנו להשתמש במסנן אליפטי (Elliptic) ממספר סיבות:

יש לו יכולת הנחתה גבוהה ב־Stop Band.

ביישומו קיבלנו סינון איכותי עם מעט מקדמים, המתאים לאילוצי חישוב וזיכרון.

להלן המפרט של המסנן הראשון:

Filter type: IIR
Response: Lowpass
Number of sections: 2
Filter order: 4
Design method: elliptic
Passband frequency: fpb1 = 0Hz→127.8076 Hz , fpb2 = 360.8691 Hz →1000 Hz
Cutoff frequency (-3dB point): fc1 = 191.04 Hz , fc2 = 322.7539 Hz
Half power frequency (-6dB point): fhp1 = 214.3224 1Hz , fhp2 = 290.0391 Hz
Stopband frequency: fsb1 = 240.8447Hz , fsb2 = 259.6436Hz
Transition band width: fsb1-fpb1 = 240.8447Hz - 127.8076 Hz = 113.0371Hz , fpb2 – fsb2 =360.8691 Hz - 259.6436 Hz =101.2255Hz
Stop-band: fsb2-fsb1 = 259.6436Hz - 240.8447Hz =18.7989Hz
Pass-band ripple: -0dB – (-1.8912dB) = 1.8912dB
Stop-band ripple: -17.2522dB

להלן המפרט של המסנן השני:

Filter type: IIR
Response: Lowpass
Number of sections: 2
Filter order: 4
Design method: elliptic
Passband frequency: fpb = 124.3896Hz
Cutoff frequency (-3dB point): fc = 141.7236Hz
Half power frequency (-6dB point): fhp= 149.4141Hz
Stopband frequency: fsb = 464.8438Hz
Transition band width: fsb-fpb = 464.8438Hz - 124.3896Hz = 340.4542Hz
Stop-band: Fsample/2 – fsb = 1000Hz - 464.8438Hz =535.1262Hz
Pass-band ripple: -0.0662dB – (-4.1139dB) = 4.0477dB
Stop-band ripple: 1dB

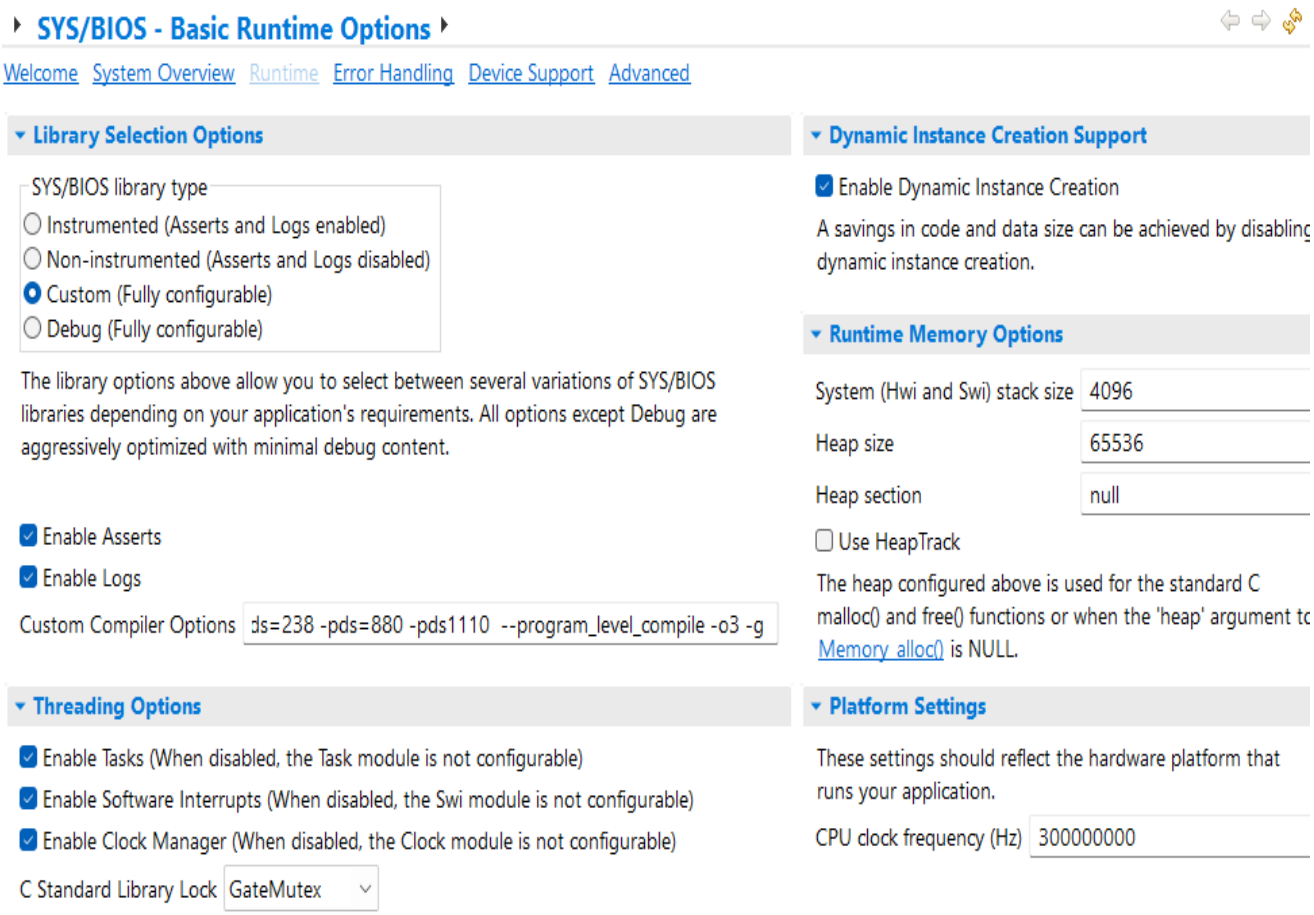
להלן המפרט של המסנן השלישי :

Filter type: IIR
Response: Highpass
Number of sections: 2
Filter order: 4
Design method: elliptic
Passband frequency: fpb = 136.1084Hz
Cutoff frequency (-3dB point): fc = 128.418Hz
Half power frequency (-6dB point): fhp= 121.82621Hz
Stopband frequency: fsb = 34.91221Hz
Transition band width: fpb -fsb = 464.8438Hz - 136.1084Hz = 328.7354Hz
Pass-band: Fsample/2 – fpb = 1000Hz - 136.1084Hz =863.8916Hz
Pass-band ripple: -0.0038dB – (-0.9996dB) = 0.9958dB
Stop-band ripple: -71.448dB

שיקולי התכנון עבור התוכנית

תצורות

BIOS

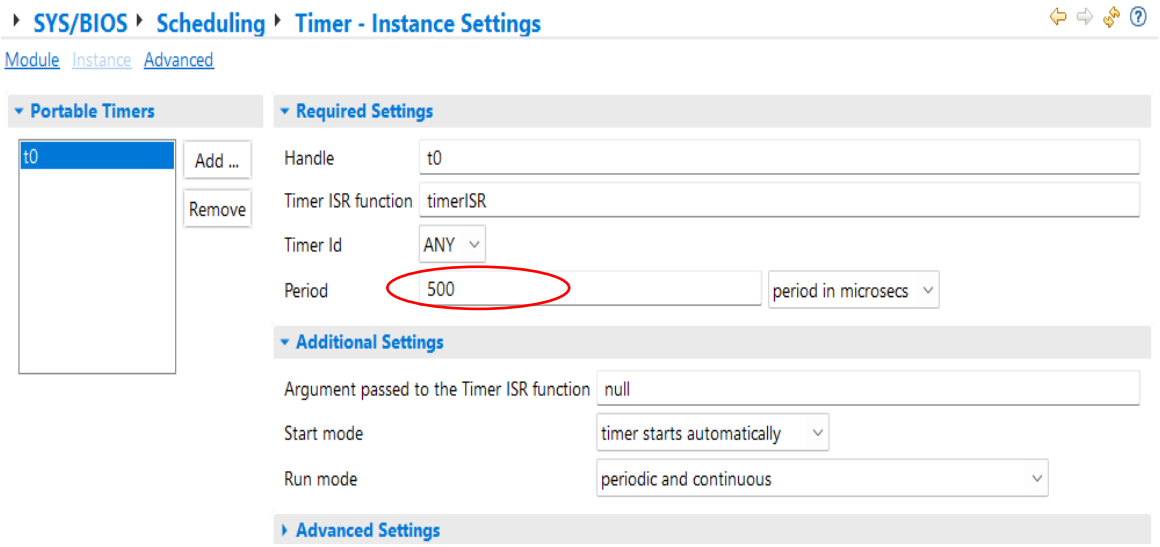


איור מס. 6 תצורת ה-BIOS

Timer

timer0 הינו מונה אשר ניתן להשתמש בו ליצירת פסיקות תקופתיות.
בפרויקט זה הוא משמש לדגום את אות של הערכים, בהתאם לתדר דגימה של 2000Hz.
הגדרנו אותו לספור בפורמט של $microsecs$ ולכן ערכו העליון הוא על פי הנוסחה:

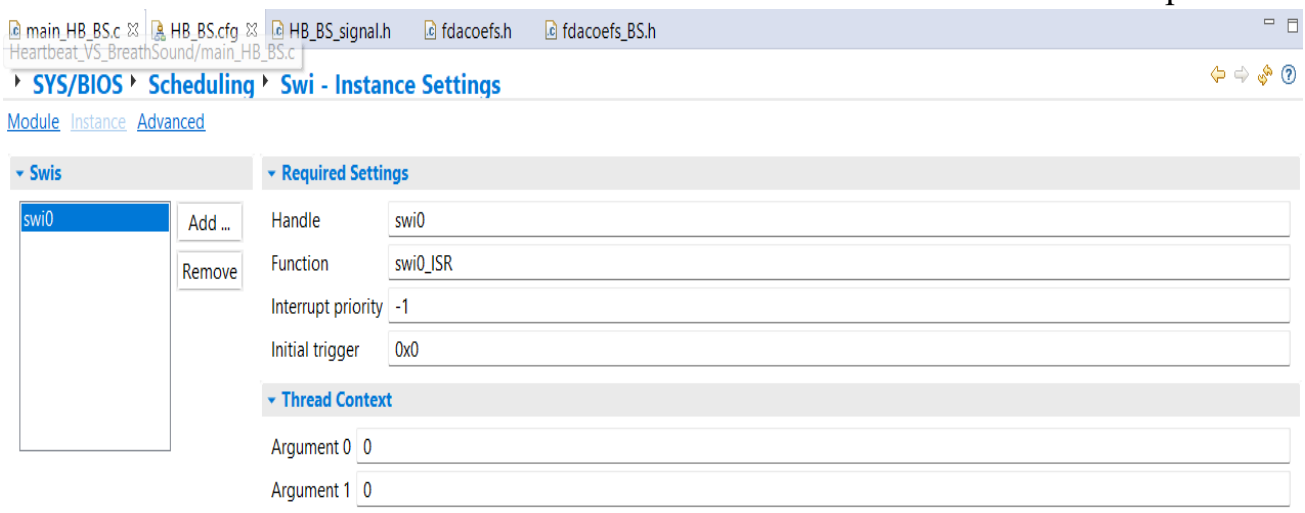
$$Timer0_{in\ microsecs} = \frac{1}{Fs} = \frac{1}{2000[Hz]} = 500\ microsecs$$



איור מס. 7 תצורת ה-Timer

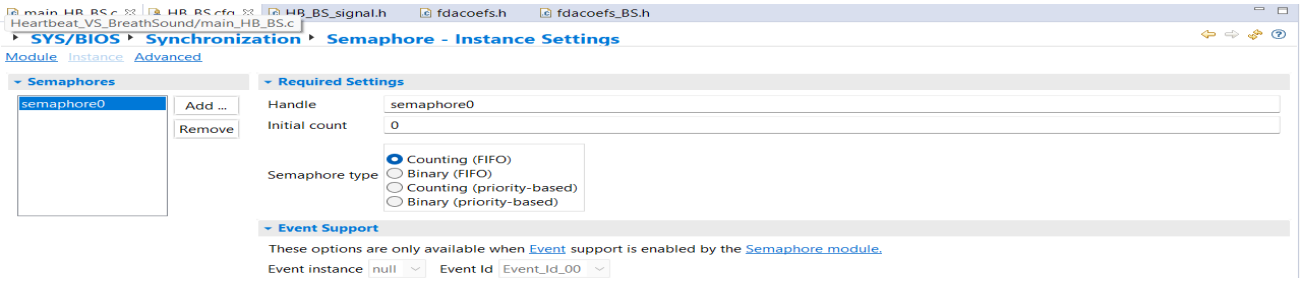
Swi

swi0 הינו פסיקת תוכנה עבור המעבד. משתמשים בו על מנת לעבור בין משימות, כך שמשימה בעלת עדיפות גבוהה יותר מקדימה משימה בעלת עדיפות נמוכה יותר. במקרה שלנו, הקריאה ל- swi0 מתרחשת בתוך שגרת הפסיקה של timer0, ב Swi אנחנו דוגמים את ה- Sample ועוברים ל Task.



איור מס. 8 תצורת ה- Swi

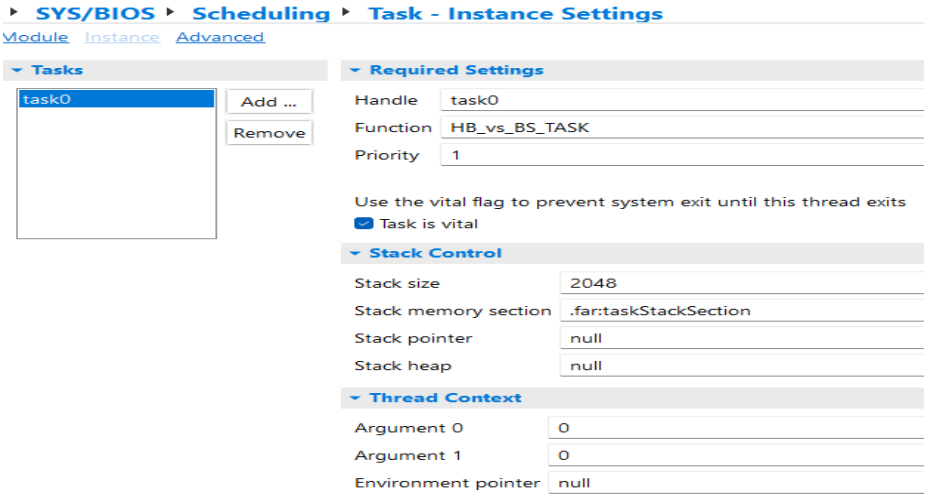
Semaphore



איור מס. 9 תצורת ה- Semaphore

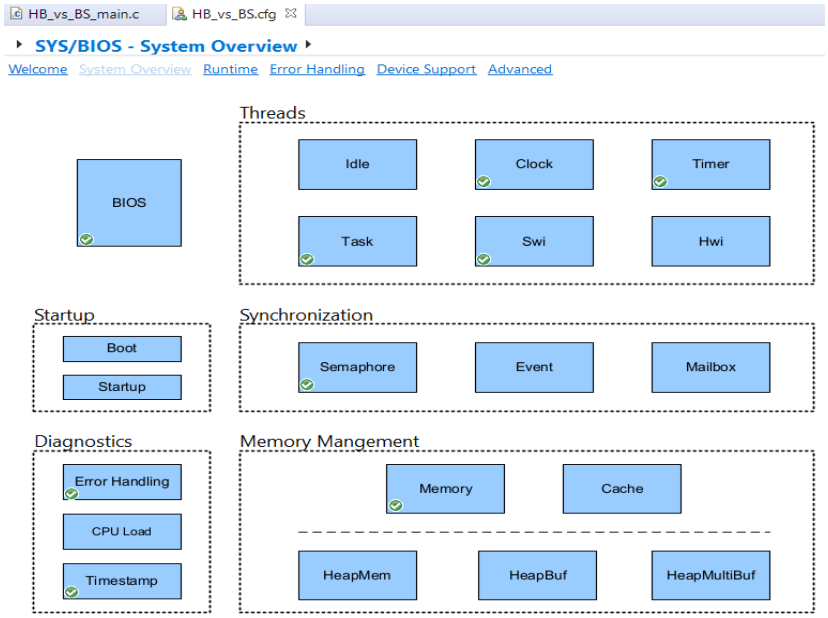
Task

זה יחידת העבודה שמבוצעת על ידי התוכנית. ניתן להשתמש בו כדי לארגן ולבצע פעולות בסדר מסוים או במרווחי זמן ספציפיים.



איור מס. 10 תצורת ה- Task

System Overview



איור מס. 11 System Overview

פונקציות התוכנית

HB_BS_Signal.h

זו קובץ header אשר מכיל מערך המייצג אות Heart Beat Breath Sound לעיבוד. המערך נקרא HB_BS_Signal והוא מכיל 4405 ערכים המייצגים את האותות.
להלן חלק מקטע הקוד:

HB_vs_BS_main.c | HB_vs_BS.cfg | HB_BS_Signal.h

```
// Heartbeat and Breath Sound input
// fs=2KHz    N=4405
#define N 4405
float HBvBS[N]={
0.0758,
0.5239,
0.6281,
0.4562,
0.0841,
-0.3104,
-0.5101,
-0.4039,
-0.0362,
0.3141,
0.4377,
0.2664,
-0.128,
-0.5201,
-0.6728,
-0.5042,
-0.1403,
0.2553,
0.4418,
0.2979,
-0.0626,
-0.4395,
-0.5585,
-0.3804,
-0.0073,
0.4013,
0.5606,
0.4014,
0.0388,
-0.3073,
-0.447,
-0.3076,
0.0851,
0.4453,
0.5647,
0.4277,
0.053,
-0.342,
-0.5164,
-0.3636,
0.0333,
0.3823,
0.5166,
```

איור מס. 12 פונקצית הספריה HB_vs_BS_signal

fdacoefs_IIR_BS.h +fdacoefs_IIR_BS.h + fdacoefs_NOTCH.h

זו קובצים header בהם מגדירים מערכים דו מימדיים המכילים את המקדמים של המסננים שתיכננו באמצעות תוכנת MATLAB.
להלן קטע הקוד:

1. IIR NOTCH filter coefficients

HB_vs_BS_main.c | HB_vs_BS.cfg | HB_BS_Signal.h | fdacoefs_NOTCH.h

```
/*
 * Filter Coefficients (C Source) generated by the Filter Design and Analysis Tool
 * Generated by MATLAB(R) 9.6 and Signal Processing Toolbox 8.2.
 * Generated on: 19-Jan-2025 13:51:13
 */

/*
 * Discrete-Time IIR Filter (real)
 * -------------------------------
 * Filter Structure    : Direct-Form II
 * Numerator Length    : 3
 * Denominator Length  : 3
 * Stable              : Yes
 * Linear Phase        : No
 */

/* General type conversion for MATLAB generated C-code  */
#include "tmwtypes.h"

/*
 * Expected path to tmwtypes.h
 * C:\Program Files\MATLAB\R2019a\extern\include\tmwtypes.h
 */
/*
 * Warning - Filter coefficients were truncated to fit specified data type.
 *   The resulting response may not match generated theoretical response.
 *   Use the Filter Design & Analysis Tool to design accurate
 *   single-precision filter coefficients.
 */
const int NL_NOTCH = 3;
const real32_T NUM_NOTCH[3] = {
     0.8259197474,   -1.168026924,   0.8259197474
};
const int DL_NOTCH = 3;
const real32_T DEN_NOTCH[3] = {
                1,   -1.168026924,   0.6518394947
};

```

2. Heart Beat IIR Lowpass filter coefficients

HB_vs_BS_main.c | HB_vs_BS.cfg | HB_BS_Signal.h | *fdacoefs_IIR_HB.h

```
 * Discrete-Time IIR Filter (real)
 * -------------------------------
 * Filter Structure    : Direct-Form II, Second-Order Sections
 * Number of Sections  : 2
 * Stable              : Yes
 * Linear Phase        : No
 */

/* General type conversion for MATLAB generated C-code  */
#include "tmwtypes.h"

#define MWSPT_NSEC 5
const int NL_IIR_HB[MWSPT_NSEC] = { 1,3,1,3,1 };
const real32_T NUM_IIR_HB[MWSPT_NSEC][3] = {
  {
     0.2371501029,               0,               0
  },
  {
                1,    1.391106009,               1
  },
  {
   0.004272235092,               0,               0
  },
  {
                1, -0.02537427098,               1
  },
  {
                1,               0,               0
  }
};
const int DL_IIR_HB[MWSPT_NSEC] = { 1,3,1,3,1 };
const real32_T DEN_IIR_HB[MWSPT_NSEC][3] = {
  {
                1,               0,               0
  },
  {
                1,   -1.701754332,   0.7476446629
  },
  {
                1,               0,               0
  },
  {
                1,   -1.728202105,   0.8940779567
  },
  {
                1,               0,               0
  }
};
```

3. Breath Sound IIR Highpass filter coefficients

HB_vs_BS_main.c | HB_vs_BS.cfg | HB_BS_Signal.h | fdacoefs_IIR_BS.h

```c
 * Discrete-Time IIR Filter (real)
 * -------------------------------
 * Filter Structure    : Direct-Form II, Second-Order Sections
 * Number of Sections  : 2
 * Stable              : Yes
 * Linear Phase        : No
 */

/* General type conversion for MATLAB generated C-code  */
#include "tmwtypes.h"

#define MWSPT_NSEC 5
const int NL_IIR_BS[MWSPT_NSEC] = { 1,3,1,3,1 };
const real32_T NUM_IIR_BS[MWSPT_NSEC][3] = {
  {
        10.4896822,                0,                0
  },
  {
                 1,    -1.998457551,                1
  },
  {
    0.04608005658,                0,                0
  },
  {
                 1,    -1.991204619,                1
  },
  {
                 1,                0,                0
  }
};
const int DL_IIR_BS[MWSPT_NSEC] = { 1,3,1,3,1 };
const real32_T DEN_IIR_BS[MWSPT_NSEC][3] = {
  {
                 1,                0,                0
  },
  {
                 1,    -1.003433347,    0.3899464309
  },
  {
                 1,                0,                0
  },
  {
                 1,    -1.723300219,    0.8929643035
  },
  {
                 1,                0,                0
  }
};
```

איור מס. 13 פונקצית הספריה fdacoefs

timer0_ISR ()

להלן קטע הקוד:

```c
/*********************************************************************
********************** Timer Interrupt (HWI) ************************
*********************************************************************/
void timerISR()
{
    /* Simply post the SWI that fetches a new sample and signals the task */
    Swi_post(swi0);
}
```

איור מס. 14 שגרת הפסיקה של ה- Timer

swi0_ISR ()

בשגרת פסיקה זו מתבצעת דגימת ערך מהמערך HB_BS_signal ושמירת הערך בתוך משתנה בשם sample.בסוף הפסיקה מתבצעת קריאה ל-semaphore לאיתות על משאבים זמינים.

להלן קטע הקוד:

```c
/*********************************************************************
********************** Software Interrupt (SWI) **********************
*********************************************************************/
void swi_ISR()
{
    /* Read the sample from the big array that merges HB & BS signals */
    sample = HBvBS[idx];

    /* Signal the main task that a new sample is ready */
    Semaphore_post(sem);
}
```

איור מס. 15 שגרת הפסיקה של Swi

HB_vs_BS_TASK

```c
/*****************************************************************
************************* Main Processing Task ********************
*****************************************************************
  1) Waits on semaphore for new data
  2) Applies Notch, HB, BS filters
  3) Computes energy, envelope, smoothing
  4) Detects half-cycles for breath/heartbeat
  5) After N=4405 samples, calculates final counts
*/
void HB_vs_BS_TASK()
{
    while (TRUE)
    {
        /* Wait for new sample to be posted by SWI */
        Semaphore_pend(sem, BIOS_WAIT_FOREVER);

        /* Apply Notch filter first */
        IIR_Biquad_NOTCH();
        /* Overwrite sample with the notch-filtered output */
        sample = IIR_filtered_arr_NOTCH[idx];

        /* Then apply HeartBeat and BreathSound filters */
        IIR_Biquad_HB();
        IIR_Biquad_BS();

        /* Compute energies (absolute values) */
        Energy_HB[idx] = (IIR_filtered_arr_HB[idx] >= 0.0f)
                       ? IIR_filtered_arr_HB[idx]
                       : -IIR_filtered_arr_HB[idx];
        Energy_BS[idx] = (IIR_filtered_arr_BS[idx] >= 0.0f)
                       ? IIR_filtered_arr_BS[idx]
                       : -IIR_filtered_arr_BS[idx];

        /* Compute envelopes */
        if (idx == 0) {
            Envelope_HB[idx] = Energy_HB[idx];
            Envelope_BS[idx] = Energy_BS[idx];
        } else {
            Envelope_HB[idx] = computeEnvelope(Energy_HB[idx], Envelope_HB[idx-1], taw);
            Envelope_BS[idx] = computeEnvelope(Energy_BS[idx], Envelope_BS[idx-1], taw);
        }

        /* Smooth the envelopes over the recent samples */
        Envelope_HB[idx] = smoothRecentValues(Envelope_HB, idx, smoothing_window);
        Envelope_BS[idx] = smoothRecentValues(Envelope_BS, idx, smoothing_window);

        /* Update the sliding windows for BS and HB */
        sliding_window_BS[window_idx_BS] = Envelope_BS[idx];
        sliding_window_HB[window_idx_HB] = Envelope_HB[idx];

        /* Detect half-cycles for breath (BS) if we've filled the window */
        if (idx >= WINDOW_SIZE_BS) {
            detectHalfCycles(sliding_window_BS, WINDOW_SIZE_BS,
                             &prev_avg_BS, &is_rising_BS, &BS_count,
                             idx, &last_transition_idx_BS);
        }

        /* Detect half-cycles for heartbeat (HB) if we've filled the window */
        if (idx >= WINDOW_SIZE_HB) {
            detectHalfCycles(sliding_window_HB, WINDOW_SIZE_HB,
                             &prev_avg_HB, &is_rising_HB, &HB_count,
                             idx, &last_transition_idx_HB);
        }

        /* Advance sliding window indices */
        window_idx_BS = (window_idx_BS + 1) % WINDOW_SIZE_BS;
        window_idx_HB = (window_idx_HB + 1) % WINDOW_SIZE_HB;

        /* If we've reached the end of the block (N=4405), compute final counts */
        if (idx == N - 1)
        {

            local_breath_count    = BS_count / 4;
            local_heartbeat_count = HB_count / 2;

            /* Reset everything for the next block */
            BS_count = 0;
            HB_count = 0;
            is_rising_BS   = 0;
            is_rising_HB   = 0;
            prev_avg_BS    = 0.0f;
            prev_avg_HB    = 0.0f;
            window_idx_BS  = 0;
            window_idx_HB  = 0;

            /* Also reset filter states to avoid carry-over */
            resetAllFilters();

            idx = 0; /* wrap around */
        }
        else
        {
            idx++;
        }
```

איור מס. 16 פונקציית HB_vs_BS_TASK()

הסבר זה נועד להציג את הפעולות העיקריות המתבצעות בפונקציה HB_vs_BS_TASK, המשמשת כ"ליבת העיבוד" של המערכת. בפונקציה זו נקלטות דגימות (Samples) של אותות לב ונשימה לאורך בלוק בגודל N (לדוגמה, 4405 דגימות), ולאחר סדרת עיבודים ופילטרים מתקבל מספר הנשימות (Breath)והפעימות (Heartbeat) באותו בלוק.

1. **המתנה לדגימה חדשה (Semaphore_pend)**
   הפונקציה רצה בלולאה אינסופית, ובכל סבב מחכה בעזרת Semaphore_pend(sem, BIOS_WAIT_FOREVER עד שה- SWI (swi_ISR) מסמן כי הגיעה דגימה חדשה. ברגע שהספירה משתחררת, המשמעות היא שיש דגימה זמינה לעיבוד.
2. **יישום מסנן Notch**
   עם קבלת הדגימה, קוראים לפונקציה IIR_Biquad_NOTCH() המסננת רעשים לא רצויים (כמו רעש רשת חשמל). התוצאה נכתבת למערך IIR_filtered_arr_NOTCH[idx] ולאחר מכן מועתקת למשתנה sample להמשך עיבוד.
3. **מסנני HeartBeat וBreathSound**
   - הפונקציה IIR_Biquad_HB() מסננת מרכיבי דופק (Heartbeat) מתוך sample ושומרת את התוצאה במערך IIR_filtered_arr_HB .
   - הפונקציה IIR_Biquad_BS() מסננת מרכיבי נשימה (BreathSound) מתוך אותו sample ושומרת את התוצאה במערך IIR_filtered_arr_BS .
4. **חישוב אנרגיה (Energy)**
   האנרגיה של כל אות מחושבת על ידי ערך מוחלט .(Absolute Value) אם הערך של IIR_filtered_arr_HBאו IIR_filtered_arr_BS שלילי, כופלים אותו ב־1− ; אחרת שומרים כמות שהוא. התוצאות נשמרות במערכים Energy_HB ו־Energy_BS.
5. **חישוב מעטפת (Envelope)**
   - בדגימה הראשונה כאשר (idx == 0), ערכי Envelope_HB ו־ Envelope_BS נקבעים ישירות לפי האנרגיה הנוכחית.
   - לאחר מכן, הפונקציה computeEnvelope() משווה את האנרגיה הנוכחית לערך המעטפת הקודם ומאפשרת עלייה מידית אך ירידה הדרגתית (עם מקדם דעיכה taw).

6. **החלקת המעטפת (Smoothing)**

כדי למנוע רעשי רקע קצרים, מפעילים פעולה של החלקה (smoothRecentValues) על המעטפת. הפונקציה משתמשת ב־ smoothing_window כדי לחשב ממוצע נע על מספר דגימות אחרונות, ובכך הופכת את האות ליציב יותר לפני זיהוי פסגות ושקעים. היא נחוצה כדי למנוע שינויים חדים מדי באות העוטף, כך שהמדידה של קצב הלב או הנשימה תהיה יציבה ומדויקת יותר.

7. **עדכון חלונות הזזה (Sliding Windows)**

לאחר ההחלקה, הערכים החדשים של Envelope_HB ו־ Envelope_BS מוכנסים לשני מערכי חלון (sliding_window_HB ו־ sliding_window_BS) כל חלון בעל גודל קבוע (למשל WINDOW_SIZE_HBו־ WINDOW_SIZE_BS) ומיועד לחישוב ממוצעים על פני מספר דגימות רצוף.

8. **זיהוי מחזורי־חצי (Half-Cycles)**

ברגע שהאינדקס idx מגיע לגודל החלון (או עובר אותו), אפשר להפעיל את detectHalfCycles , הבודקת אם הממוצע הנוכחי של החלון (sliding_window_HB או sliding_window_BS) עולה או יורד ביחס לממוצע הקודם.

- מעבר ממצב עולה למצב יורד (או להפך) נספר כ־ Half-Cycle אחד, ומתווסף למונה HB_count (לפעימות לב) או BS_count (לנשימה).

9. **עדכון אינדקסי החלון (window_idx_BS, window_idx_HB)**

בכל לולאה, האינדקסים של החלונות מוזזים ב־1 (בחישוב מודולרי), כך שהחלון תמיד כולל את הדגימות האחרונות ביותר.

10. **בדיקה אם הגענו לסוף הבלוק (N דגימות)**

אם idx == (N - 1) (למשל 4405 דגימות), המשמעות היא שסיימנו לעבד בלוק שלם:

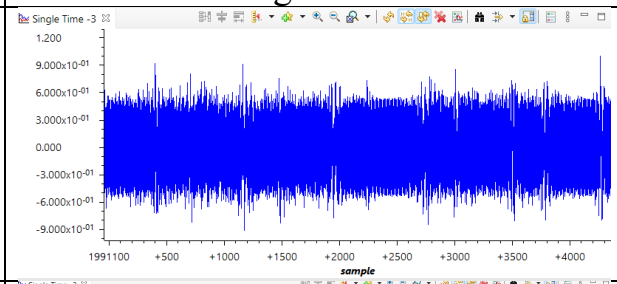
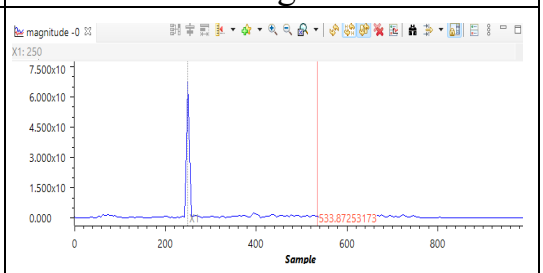
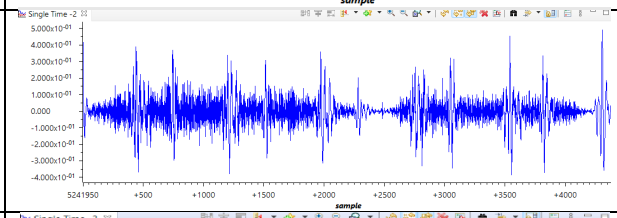
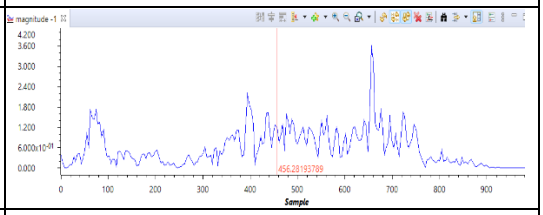
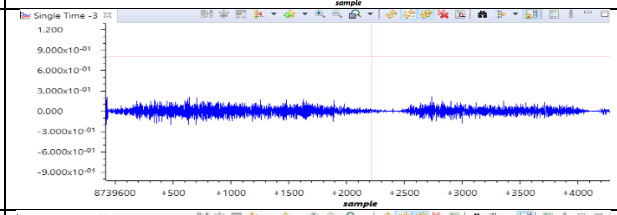
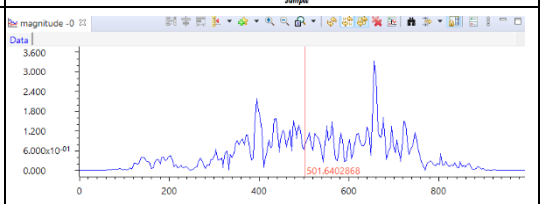
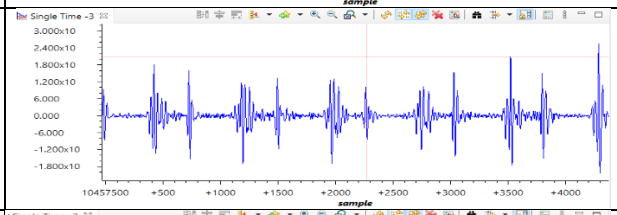
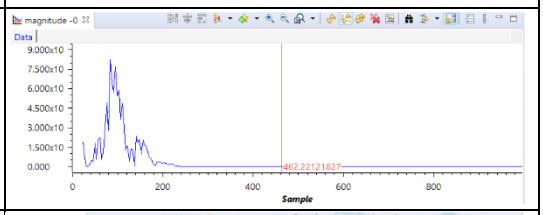
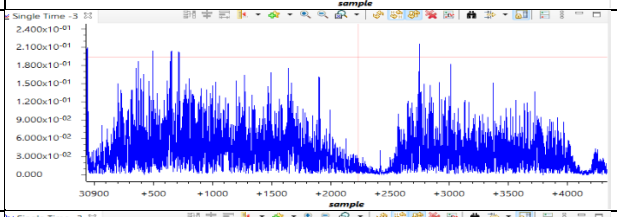
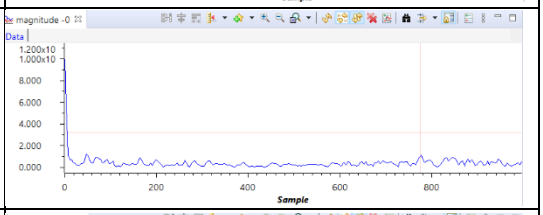
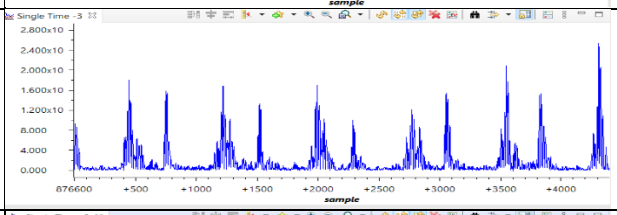
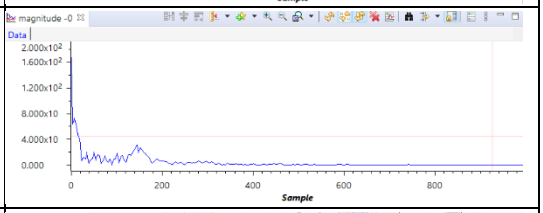
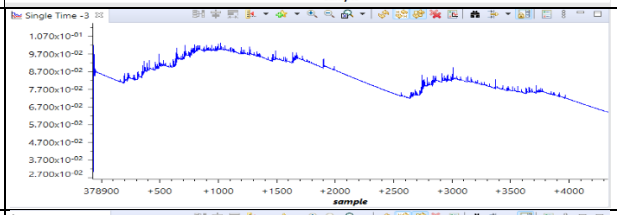
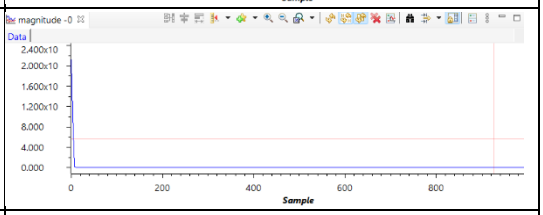
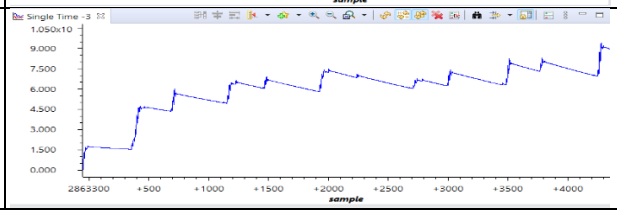
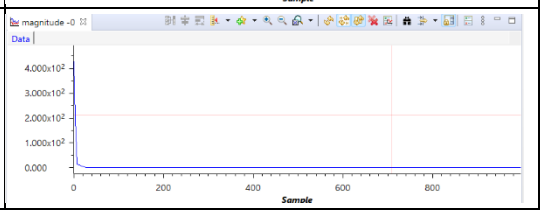
- מחשבים את כמות הנשימות: local_breath_count = BS_count / 4 . חלוקה ב־4 כי עבור נשימה מלאה (שאיפה + נשיפה) מזוהים בפועל 4 מעברים (פסגות/שקעים).
- מחשבים את כמות הפעימות: local_heartbeat_count = HB_count / 2 . חלוקה ב־2 כי כל פעימת לב מורכבת משני חצאי־מחזור (פסגה ושפל).
- מאפסים את המונים, הדגלים והמצבים הרלוונטיים (BS_count, HB_count, is_rising_BS is_rising_HB, prev_avg_BS, prev_avg_HB) ואת מצב המסננים (resetAllFilters) .
- לבסוף מגדירים idx = 0 כדי להתחיל עיבוד בלוק חדש.

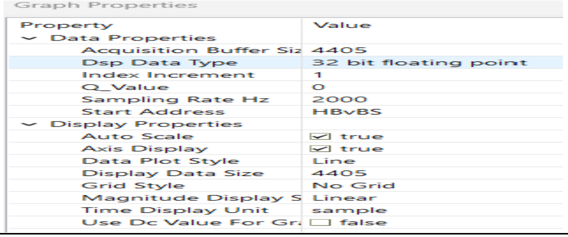
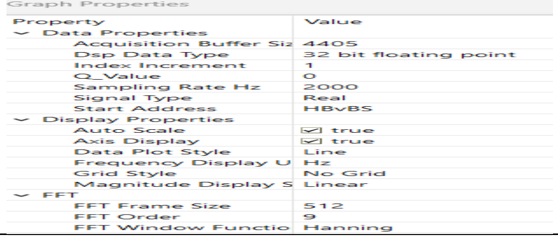
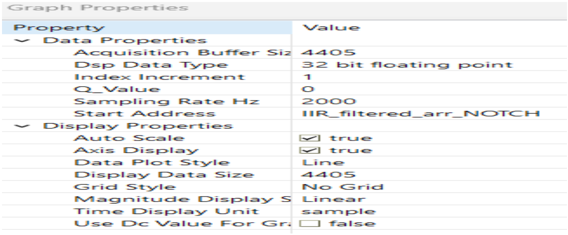
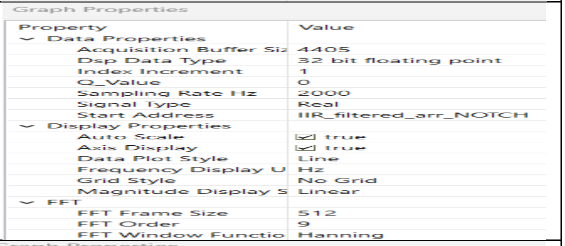
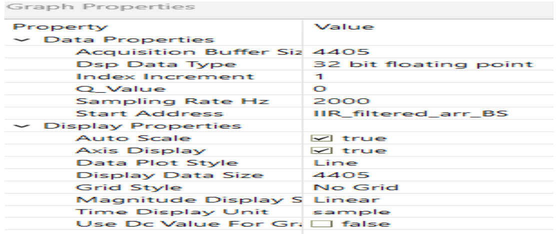
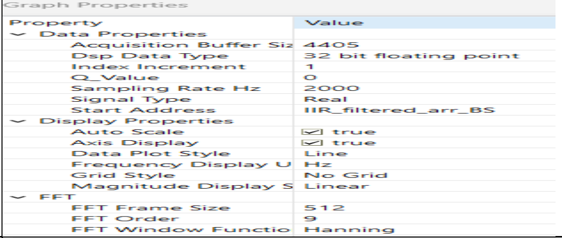
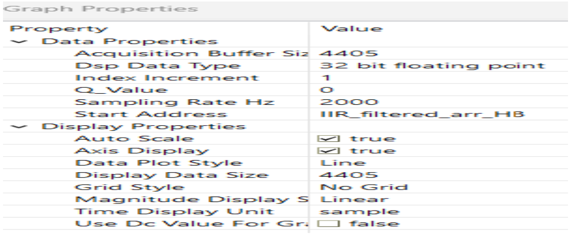
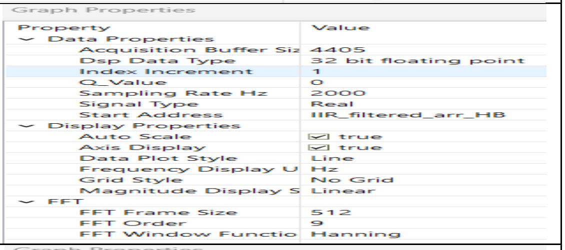
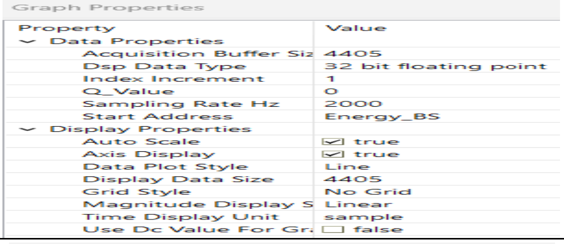
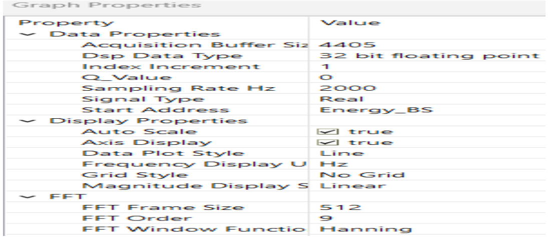
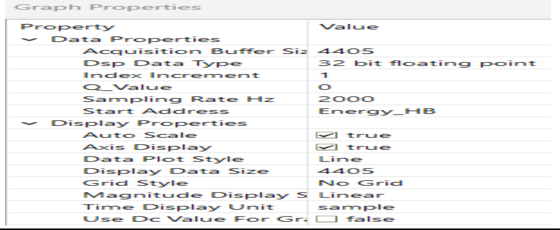
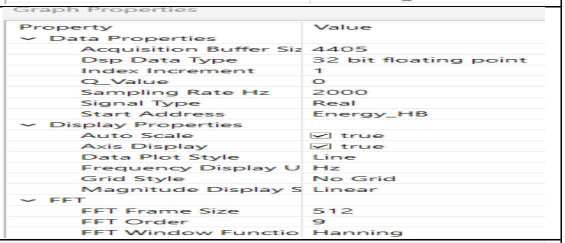
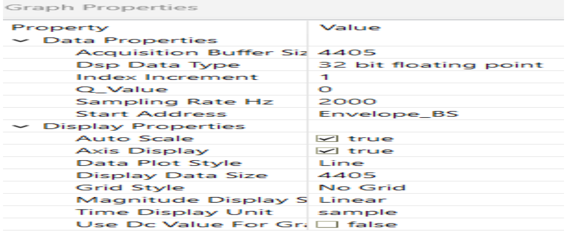
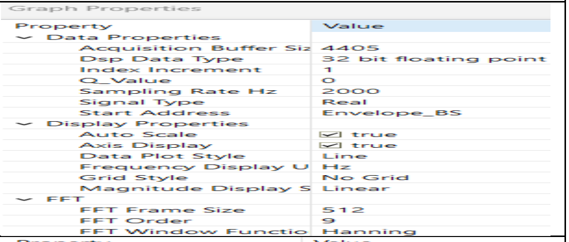
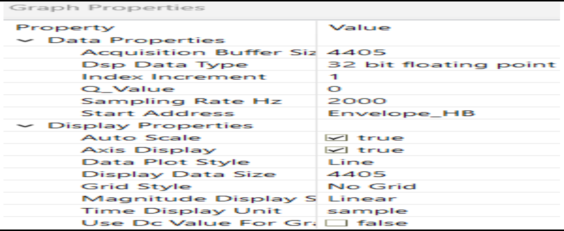
11. **אחרת (אם לא הגענו ל־ Nדגימות)**

פשוט מגדילים את idx ב־1 וממשיכים בלולאה עם הדגימה הבאה.

זהו סדר הפעולות בפונקציה המרכזית HB_vs_BS_TASK, שמטרתה לסנן ולזהות בצורה אמינה את מספר הנשימות ופעימות הלב בכל מקבץ (Buffer) של דגימות.

תוצאות ההרצה

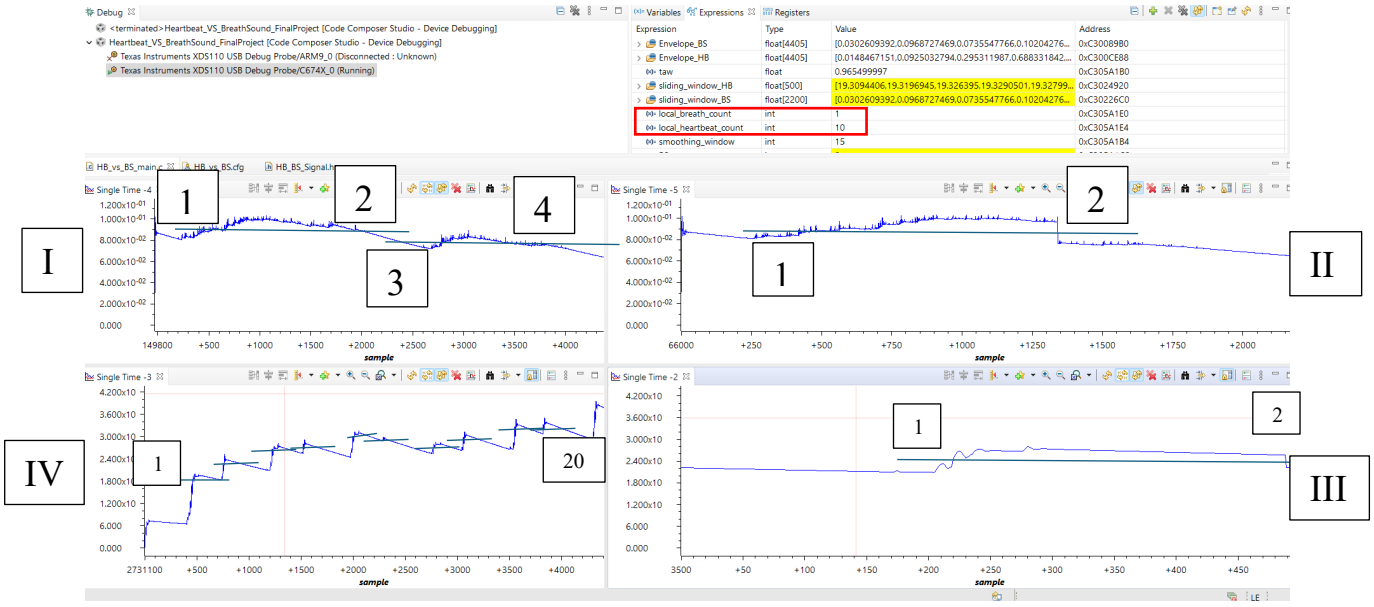
Graph	Single Time	FFT Magnitude
HBvBS		
IIR_filtered_ arr_NOTCH		
IIR_filtered_ arr_BS		
IIR_filtered_ arr_HB		
Energy_BS		
Energy_HB		
Envelope_BS		
Envelope_HB		

Graph Properties	Single Time	FFT Magnitude
HBvBS	 Graph Properties Property Value Data Properties Acquisition Buffer Siz 4405 Dsp Data Type 32 bit floating point Index Increment 1 Q_Value 0 Sampling Rate Hz 2000 Start Address HBvBS Display Properties Auto Scale ☑ true Axis Display ☑ true Data Plot Style Line Display Data Size 4405 Grid Style No Grid Magnitude Display S Linear Time Display Unit sample Use Dc Value For Gr. ☐ false	 Graph Properties Property Value Data Properties Acquisition Buffer Siz 4405 Dsp Data Type 32 bit floating point Index Increment 1 Q_Value 0 Sampling Rate Hz 2000 Signal Type Real Start Address HBvBS Display Properties Auto Scale ☑ true Axis Display ☑ true Data Plot Style Line Frequency Display U Hz Grid Style No Grid Magnitude Display S Linear FFT FFT Frame Size 512 FFT Order 9 FFT Window Functio Hanning
IIR_filtered_ arr_NOTCH	 Graph Properties Property Value Data Properties Acquisition Buffer Siz 4405 Dsp Data Type 32 bit floating point Index Increment 1 Q_Value 0 Sampling Rate Hz 2000 Start Address IIR_filtered_arr_NOTCH Display Properties Auto Scale ☑ true Axis Display ☑ true Data Plot Style Line Display Data Size 4405 Grid Style No Grid Magnitude Display S Linear Time Display Unit sample Use Dc Value For Gr. ☐ false	 Graph Properties Property Value Data Properties Acquisition Buffer Siz 4405 Dsp Data Type 32 bit floating point Index Increment 1 Q_Value 0 Sampling Rate Hz 2000 Signal Type Real Start Address IIR_filtered_arr_NOTCH Display Properties Auto Scale ☑ true Axis Display ☑ true Data Plot Style Line Frequency Display U Hz Grid Style No Grid Magnitude Display S Linear FFT FFT Frame Size 512 FFT Order 9 FFT Window Functio Hanning
IIR_filtered_ arr_BS	 Graph Properties Property Value Data Properties Acquisition Buffer Siz 4405 Dsp Data Type 32 bit floating point Index Increment 1 Q_Value 0 Sampling Rate Hz 2000 Start Address IIR_filtered_arr_BS Display Properties Auto Scale ☑ true Axis Display ☑ true Data Plot Style Line Display Data Size 4405 Grid Style No Grid Magnitude Display S Linear Time Display Unit sample Use Dc Value For Gr. ☐ false	 Graph Properties Property Value Data Properties Acquisition Buffer Siz 4405 Dsp Data Type 32 bit floating point Index Increment 1 Q_Value 0 Sampling Rate Hz 2000 Signal Type Real Start Address IIR_filtered_arr_BS Display Properties Auto Scale ☑ true Axis Display ☑ true Data Plot Style Line Frequency Display U Hz Grid Style No Grid Magnitude Display S Linear FFT FFT Frame Size 512 FFT Order 9 FFT Window Functio Hanning
IIR_filtered_ arr_HB	 Graph Properties Property Value Data Properties Acquisition Buffer Siz 4405 Dsp Data Type 32 bit floating point Index Increment 1 Q_Value 0 Sampling Rate Hz 2000 Start Address IIR_filtered_arr_HB Display Properties Auto Scale ☑ true Axis Display ☑ true Data Plot Style Line Display Data Size 4405 Grid Style No Grid Magnitude Display S Linear Time Display Unit sample Use Dc Value For Gr. ☐ false	 Graph Properties Property Value Data Properties Acquisition Buffer Siz 4405 Dsp Data Type 32 bit floating point Index Increment 1 Q_Value 0 Sampling Rate Hz 2000 Signal Type Real Start Address IIR_filtered_arr_HB Display Properties Auto Scale ☑ true Axis Display ☑ true Data Plot Style Line Frequency Display U Hz Grid Style No Grid Magnitude Display S Linear FFT FFT Frame Size 512 FFT Order 9 FFT Window Functio Hanning
Energy_BS	 Graph Properties Property Value Data Properties Acquisition Buffer Siz 4405 Dsp Data Type 32 bit floating point Index Increment 1 Q_Value 0 Sampling Rate Hz 2000 Start Address Energy_BS Display Properties Auto Scale ☑ true Axis Display ☑ true Data Plot Style Line Display Data Size 4405 Grid Style No Grid Magnitude Display S Linear Time Display Unit sample Use Dc Value For Gr. ☐ false	 Graph Properties Property Value Data Properties Acquisition Buffer Siz 4405 Dsp Data Type 32 bit floating point Index Increment 1 Q_Value 0 Sampling Rate Hz 2000 Signal Type Real Start Address Energy_BS Display Properties Auto Scale ☑ true Axis Display ☑ true Data Plot Style Line Frequency Display U Hz Grid Style No Grid Magnitude Display S Linear FFT FFT Frame Size 512 FFT Order 9 FFT Window Functio Hanning
Energy_HB	 Graph Properties Property Value Data Properties Acquisition Buffer Siz 4405 Dsp Data Type 32 bit floating point Index Increment 1 Q_Value 0 Sampling Rate Hz 2000 Start Address Energy_HB Display Properties Auto Scale ☑ true Axis Display ☑ true Data Plot Style Line Display Data Size 4405 Grid Style No Grid Magnitude Display S Linear Time Display Unit sample Use Dc Value For Gr. ☐ false	 Graph Properties Property Value Data Properties Acquisition Buffer Siz 4405 Dsp Data Type 32 bit floating point Index Increment 1 Q_Value 0 Sampling Rate Hz 2000 Signal Type Real Start Address Energy_HB Display Properties Auto Scale ☑ true Axis Display ☑ true Data Plot Style Line Frequency Display U Hz Grid Style No Grid Magnitude Display S Linear FFT FFT Frame Size 512 FFT Order 9 FFT Window Functio Hanning
Envelope_BS	 Graph Properties Property Value Data Properties Acquisition Buffer Siz 4405 Dsp Data Type 32 bit floating point Index Increment 1 Q_Value 0 Sampling Rate Hz 2000 Start Address Envelope_BS Display Properties Auto Scale ☑ true Axis Display ☑ true Data Plot Style Line Display Data Size 4405 Grid Style No Grid Magnitude Display S Linear Time Display Unit sample Use Dc Value For Gr. ☐ false	 Graph Properties Property Value Data Properties Acquisition Buffer Siz 4405 Dsp Data Type 32 bit floating point Index Increment 1 Q_Value 0 Sampling Rate Hz 2000 Signal Type Real Start Address Envelope_BS Display Properties Auto Scale ☑ true Axis Display ☑ true Data Plot Style Line Frequency Display U Hz Grid Style No Grid Magnitude Display S Linear FFT FFT Frame Size 512 FFT Order 9 FFT Window Functio Hanning
Envelope_HB	 Graph Properties Property Value Data Properties Acquisition Buffer Siz 4405 Dsp Data Type 32 bit floating point Index Increment 1 Q_Value 0 Sampling Rate Hz 2000 Start Address Envelope_HB Display Properties Auto Scale ☑ true Axis Display ☑ true Data Plot Style Line Display Data Size 4405 Grid Style No Grid Magnitude Display S Linear Time Display Unit sample Use Dc Value For Gr. ☐ false	 Property Value Data Properties Acquisition Buffer Siz 4405 Dsp Data Type 32 bit floating point Index Increment 1 Q_Value 0 Sampling Rate Hz 2000 Signal Type Real Start Address Envelope_HB Display Properties Auto Scale ☑ true Axis Display ☑ true Data Plot Style Line Frequency Display U Hz Grid Style No Grid Magnitude Display S Linear FFT FFT Frame Size 512 FFT Order 9 FFT Window Functio Hanning

תוצאת ה Count

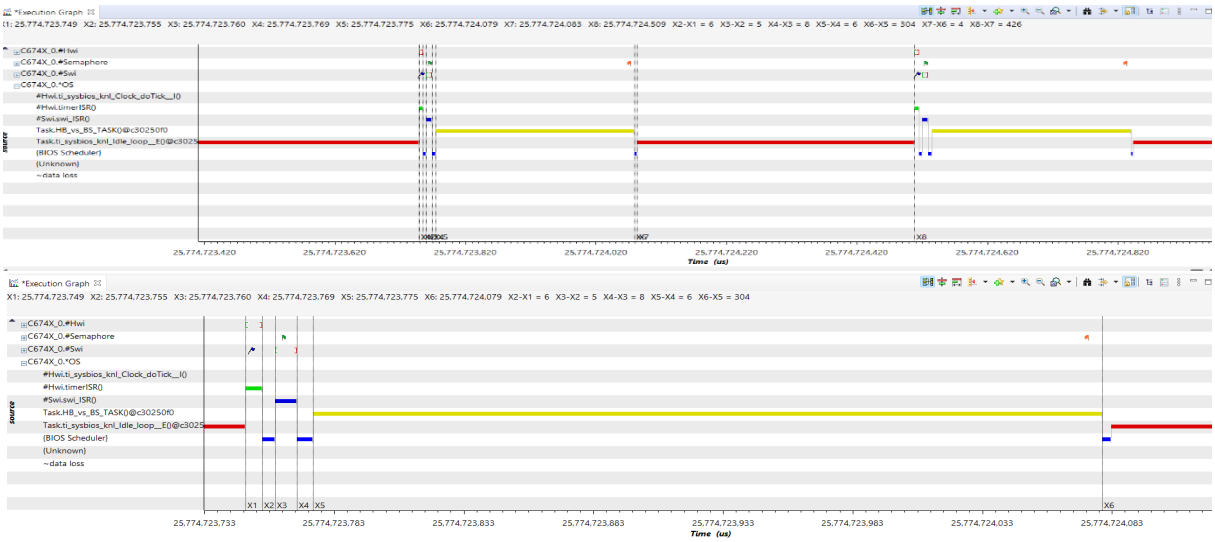
בתרשים שלפניכם מוצגות שתי מעטפות – האחת לאות הנשימה (Envelope_BS) והשנייה לאות הדופק (Envelope_HB).בנוסף, ניתן לראות את חלונות ההזזה (sliding windows) המשמשים לחישוב הממוצע ולזיהוי הפסגות והשקעים.

לדוגמה, בתרשים הנשימה (BS) זוהו 4 מעברים (חצאי־מחזור), שבעקבותיהם הספירה מתורגמת לנשימה אחת שלמה כמו באיור למטה ממעבר מאיור I -> II . בתרשים הדופק (HB) נצפו 20 מעברים, אשר בהתאם לאלגוריתם מתורגמים ל־10 פעימות לב שלמות כמו באיור למטה ממעבר מאיור III -> IV .



Execution graph, Task and CPU load

Execution graph

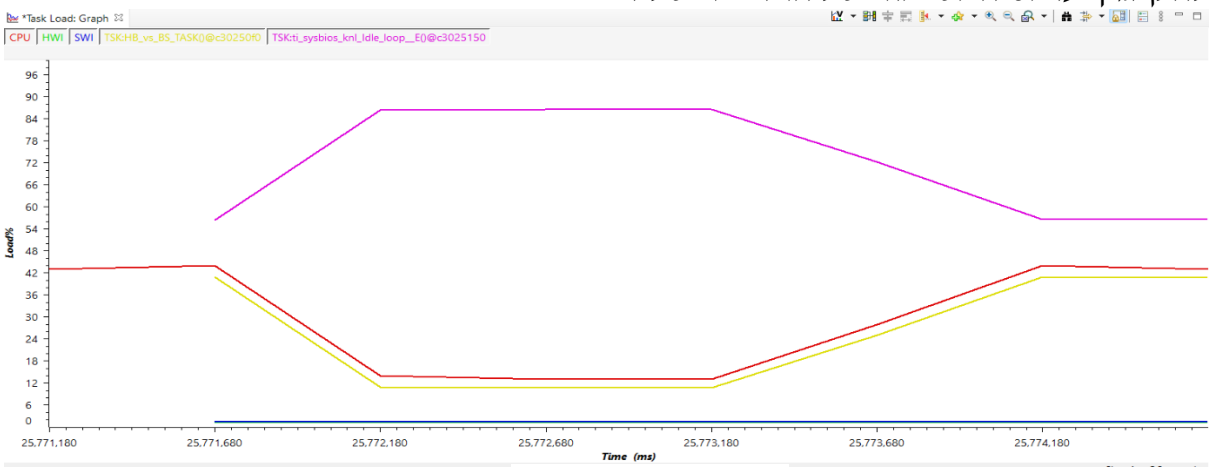


תוצאות :

Hwi.timerISR() = 6 [us]
Swi. ISR() = 8 [us]
Task.HB_vs_BS_TASK() = 304 [us]
Task.ti_sysbios_knl_idle_loop() = 426 [us]

Task load

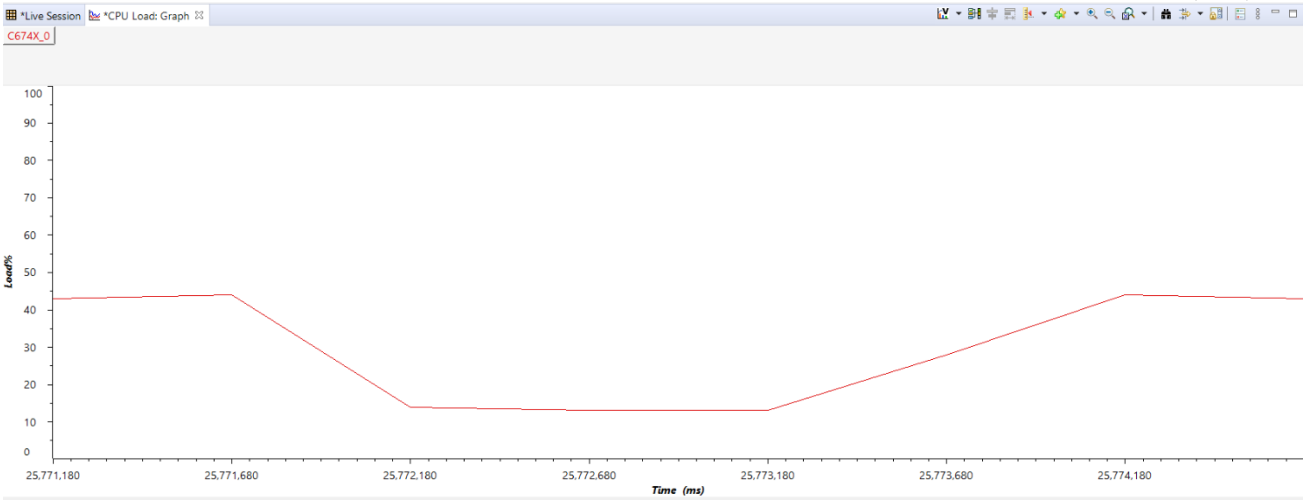
גרף ה- Task load (גרף עומס המשימות) מציג באופן חזותי את עומס העבודה של התוכנית.
להלן גרף עומס המשימות של התוכנית שלנו :



איור מס.29 Task load

CPU load

גרף ה-CPU load (עומס המעבד) מציג באופן חזותי את ניצול כוח העיבוד של המעבד.
עומס גבוה מציין שהמעבד נמצא בשימוש רב ועומס נמוך מציין שהמעבד אינו בשימוש רב.
להלן גרף עומס המעבד של התוכנית שלנו :



איור מס.30 CPU load

- **<u>הערות חשובות:</u>**
  1. המערכת שבנינו נוכל לוטון אותה בכל מני ערכים של דופק לב וקצב נשימה אפילו שונים ממה שהכנסנו אך התנאי היחידי הוא להכניס נתונים שמתאימים לתדרי הפילטרים שהשתמשנו כלומר אנו יכולים להפוך את סדר הכנסת הנתונים להעביר בהתחלה אות לקצב נשימה ואחרי זה הפרעה ואז קצב דופק לב ויחד עם הפילטרים מבודדים את כל מני אותות.
  2. בקוד שלנו כתבנו מגוון פונקציות שבהן השתמשנו בתוך **HB_vs_BS_TASK**, במטרה לארגן את הקוד ולהקל על הבנתו. אנו מודעים לכך שביצוע הקוד באיטרציה אחת, ללא קריאות לפונקציות חיצוניות, עשוי להיות מהיר יותר ולפגוע פחות בביצועי המערכת, אך בחרנו בכל זאת להשתמש בפונקציות כדי לשמור על קריאות וברור הקוד. מכיוון שזהו מיני-פרויקט, החלטנו להעדיף נוחות קריאה ותחזוקה על פני אופטימיזציה מוחלטת של הביצועים.