אוניברסיטת בן גוריון בנגב

הפקולטה למדעי ההנדסה

המחלקה להנדסת חשמל ומחשבים

מבנה מחשבים ספרתיים

פרויקט סופי : בקר טמפרטורה מבוסס PID

אמיר מלאק 203764683

משה בנסימון 300779022

22.01.2015

1. הגדרת הפרויקט

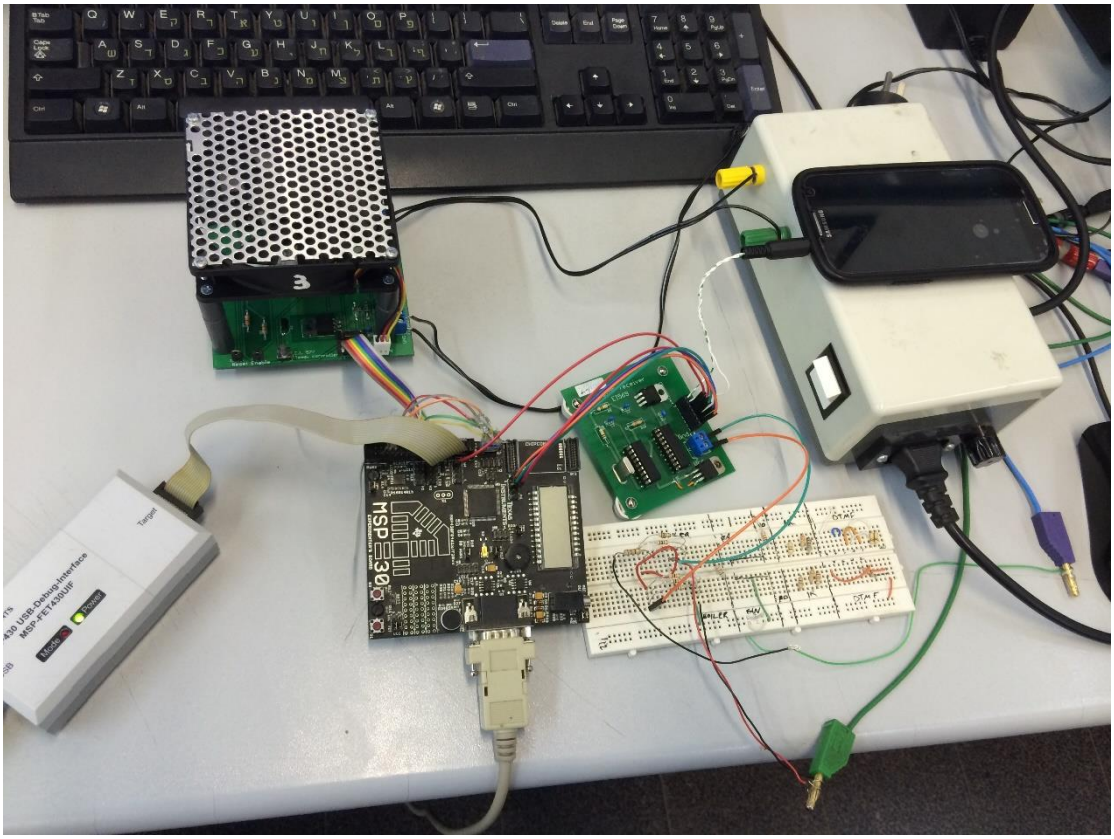
בפרויקט זה מימשנו בקר טמפרטורה ממוחשב המבוסס על בקרת PID .

המשתמש מכניס ארבע טמפרטורות אליהן הוא רוצה להגיע, והמערכת שלנו מגיעה לטמפרטורות אלה במרווחי זמן שווים. באם המערכת מגיעה לטמפרטורה הרצויה, היא מבקרת על הטמפרטורה (בקרה שמבוססת PID) בכך שהיא מפעילה מאוורר (קירור) או נגד הספק (חימום) בהתאמה ומאפשרת לטמפרטורת הבקר להישאר קבועה סביב טמפרטורת היעד בתוך "שרוול" של סטייה מוגדרת מראש.

בפרויקט נעשה שימוש ב DTMF(dual tone multi frequency), שזוהי מערכת שיכולה לזהות טונים שונים של פלאפון ולתרגמם לאותות בהתאם למספר הנלחץ.

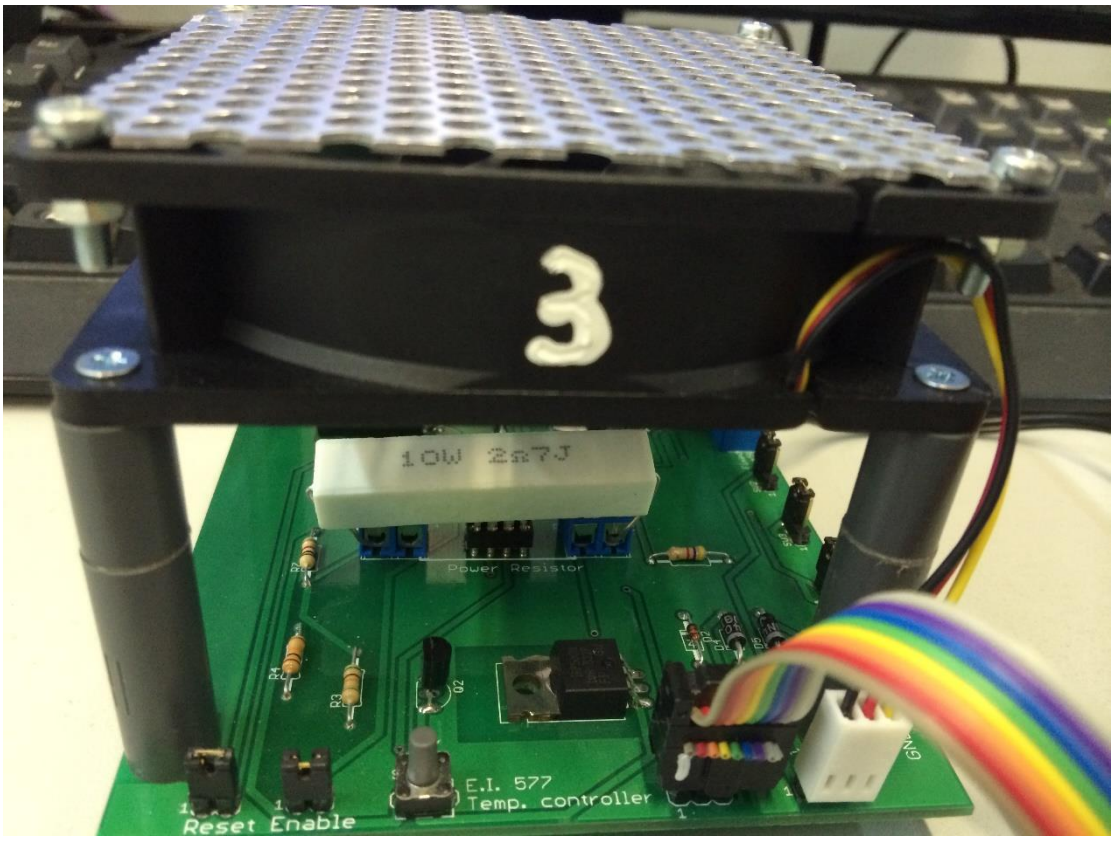
המשתמש יכול לבחור באיזו דרך הוא רוצה להזין את הטמפרטורות הרצויות. דרך ה PC או דרך הפלאפון.

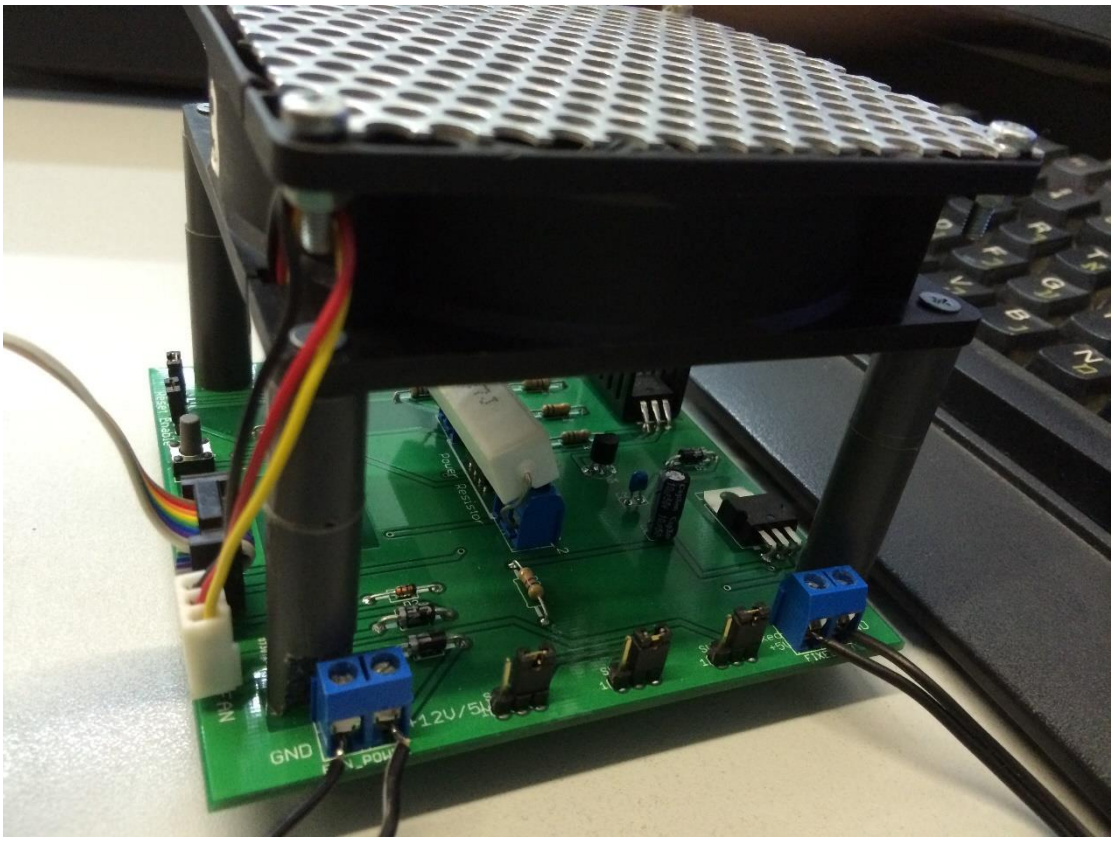
2. תיאור המערכת



המערכת כוללת שימוש בשני חלקים עיקריים : בקר טמפרטורה, ו DTMF .

2.1. תיאור בקר הטמפרטורה





בקר הטמפרטורה מורכב משלושה חלקים עיקריים :

1. מאוורר.

2. נגד הספק של 2Ω.

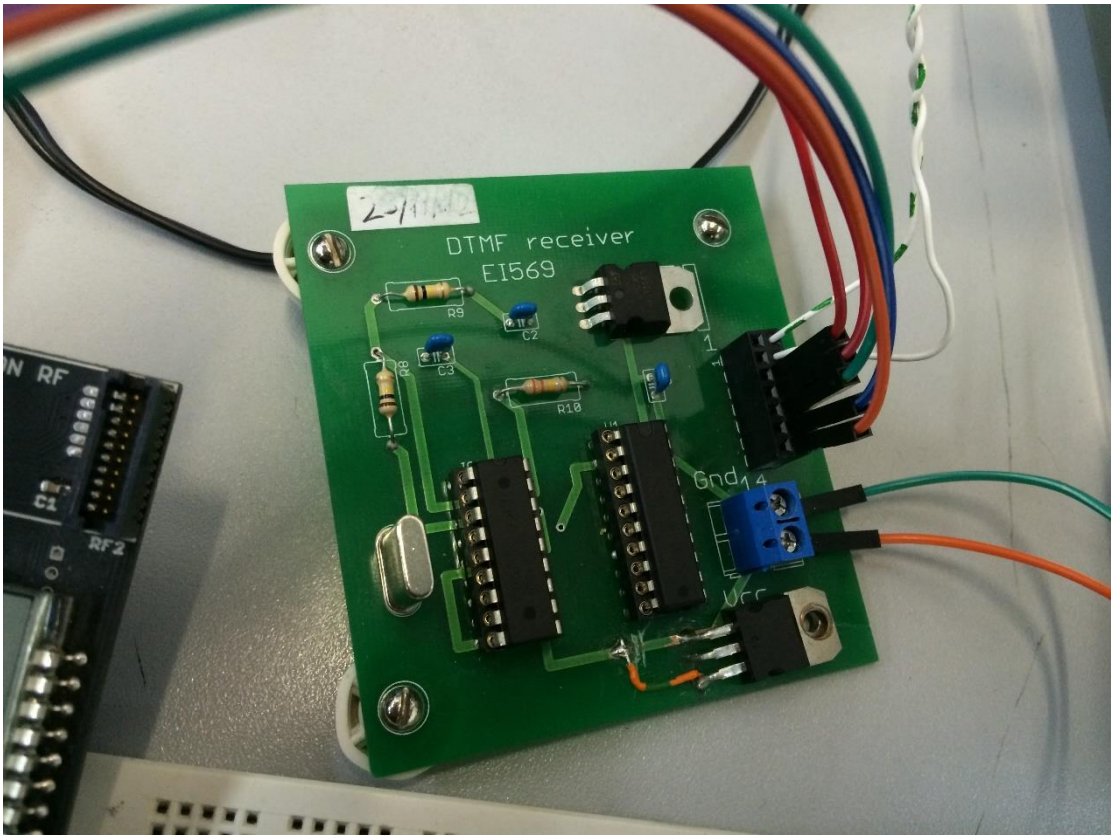
3. חיישן טמפרטורה מסוג DS1621 .

ע"מ להפעיל את המאוורר נדרש לספק מתח של 12V, וע"מ להפעיל את הנגד הספק יש לספק מתח של 5V .

הפעלת המאוורר והנגד הספק מתבצעת באמצעות מתן "0" בפין המתאים, וכיבויים מתבצע ע"י מתן "1".

הקריאה מחיישן הטמפרטורה מתבצעת באמצעות תקשורת טורית I2C בפרוטוקול התקשורת המתאים.

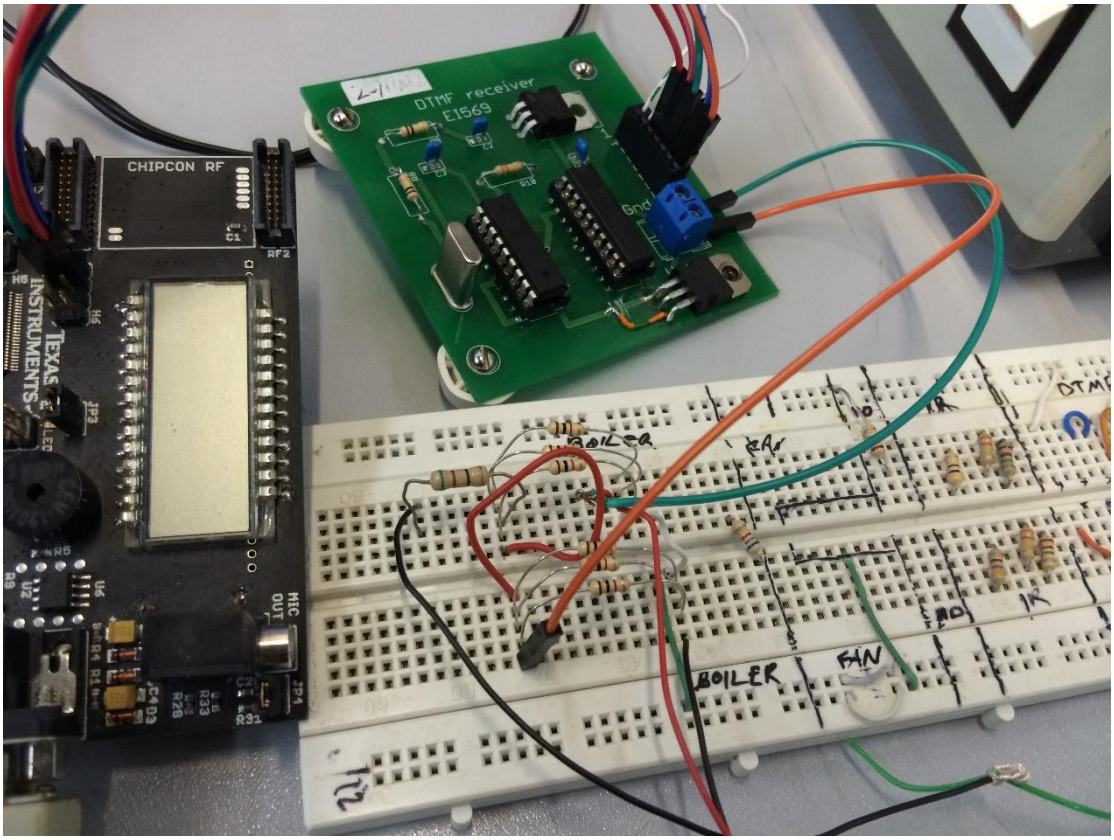
2.2. תיאור ה DTMF



רכיב זה מקבל טונים של פלאפון ויודע לזהותם ולהמירם לביטים בהתאם למספר שהוקש. ע"י מתן פסיקה בפורט המתאים, ה DTMF מעדכן את הערך החדש של ההקשה ושולח את הביטים לבקר.

ה DTMF דורש מתח של 6V ע"מ לפעול כראוי, והספק שניתן לנו הינו ספק של 9V. לכן, נאלצנו לבנות מחלק מתח מתאים כך שיסופק המתח הנדרש לכרטיס.

בהתחשב בהתנגדות הפנימית של ה DTMF, בנינו את המעגל החשמלי הבא :



מעגל זה מייצג מחלק מתח אשר מספק מתח רצוי של 6.5V . המעגל מורכב מנגד אחד של 51Ω, בטור לשישה נגדים של 1KΩ המחוברים במקביל אחד לשני והיוצרים נגד שקול של 166Ω.

בעזרת חישוב מתאים ניתן לראות כי אכן מסופק המתח הרצוי.

3. תיאור הפורטים

3.1. Port 3 : פורט זה משמש לתקשורת עם בקר הטמפרטורה בעזרת פרוטוקול תקשורת I2C, ובין היתר לשליטה על המאוורר ועל הנגד הספק.

P3.0 : זהו פין עבור TOUT .

P3.1 : זהו פין עבור SDA(serial data) .

P3.2 : זהו פין עבור SCL(serial clock) .

P3.4 : זהו פין עבור שליטה במאוורר.

P3.5 : זהו פין עבור שליטה בנגד הספק .

3.2. Port 2 : פורט זה משמש עבור רגל הפסיקה של ה DTMF .

P2.7 : זהו פין עבור ה interrupt של ה DTMF .

3.3. Port 7 : פורט זה משמש לקבלת הביטים הנשלחים מה DTMF בהתאם למספר שנלחץ בפלאפון.

P7.0~P7.3: אלה הפינים עבור ארבעת הביטים הנשלחים מה DTMF .

4. תיאור הטיימרים

בפרויקט שלנו עשינו שימוש בשני טיימרים עיקריים : Timer A, Timer B .

4.1. Timer A : טיימר זה פוסק כל שנייה, וברוטינה שלו מתבצע החישוב של בקרת ה PID על הטמפרטורה, ובהתאם מתקבלת החלטה להפעיל או לכבות את המאוורר או את נגד הספק בהתאמה.

4.2. Timer B : טיימר זה פוסק כל שנייה, וברוטינה שלו מתבצעת הבדיקה אם חלפו שתי דקות מאז עדכון טמפרטורת היעד. באם חלפו שתי דקות, מתבצע עדכון של טמפרטורה היעד החדשה והתוכנית רצה מחדש ומבקרת על הטמפרטורה החדשה.

5. שימוש בתקשורת RS232

בפרויקט נעשה שימוש בפרוטוקול תקשורת RS232, באמצעות תקשורת זו נשלחות הטמפרטורות הרצויות אותן המשתמש הכניס לבקר ה MSP . ובנוסף, קריאת הטמפרטורה העכשווית נשלחת ל PC מהבקר בעזרת תקשורת זו.

תקשורת I2C

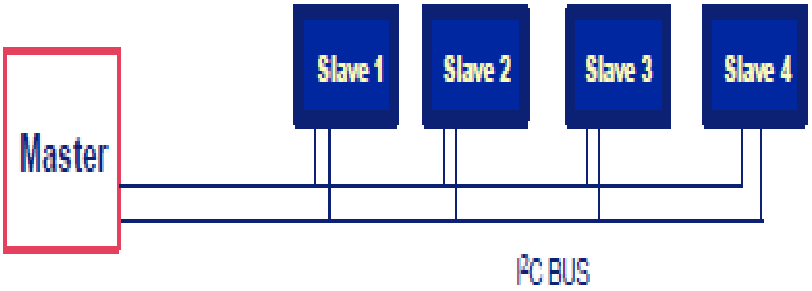
תקשורת I^2C היא תקשורת טורית בין מעבד- MASTER ורכיב עבד – SLAVE .

על פס תקשורת I^2C יכולים להתחבר מספר רכיבים שונים (זיכרונות , ממירים, שעוני זמן אמת וכו'). הרכיב המנהל את תהליך התקשורת (המעבד) נקרא MASTER והרכיבים המתחברים אליו נקראים SLAVES . בתקשורת זו ישנם שני קווים, קו הנתונים הטורי - SDA שהוא דו כיווני וקו השעון הטורי - SCL שהוא חד כיווני ומופעל על ידי ה MASTER .

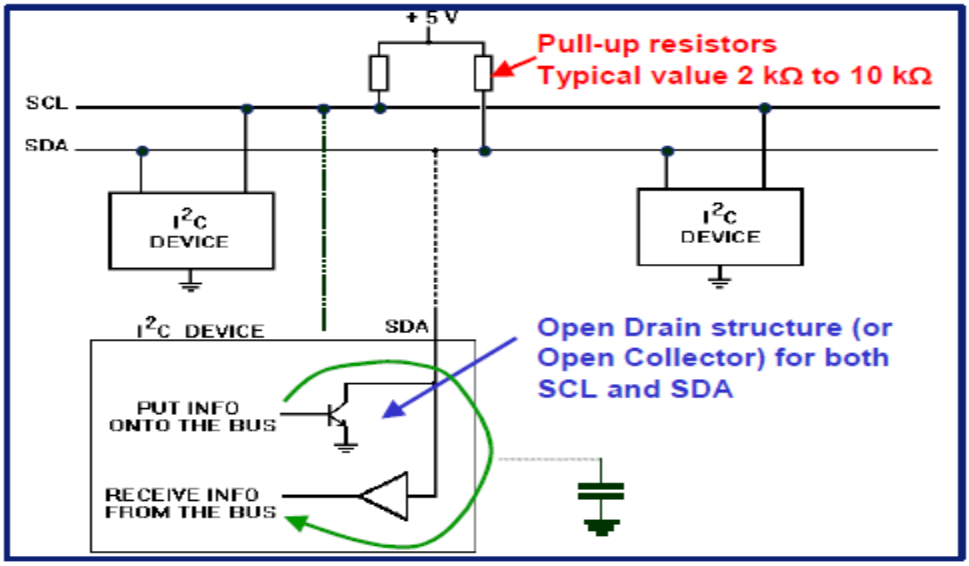
בנוסף, ה MASTER שולט על הגישה לפס ויוצר את מצבי ה START (התחלה) וה STOP (סיום).

הערה : על המשתמש להגדיר את הכיוונים של הפינים SDA ו SCL דרך $TRISC < 4:3 > bits$.

איור 1 א' - חיבור של מספר רכיבי SLAVE אל ה MASTER על קו התקשורת I^2C



באיור הנ"ל ניתן לראות 4 רכיבי SLAVE המתחברים אל MASTER. באיור 1 ב' יש פרוט של נגדי ה Pull – up וכיצד נראית דרגת היציאה והכניסה של רכיב המתחבר בתקשורת I^2C .



איור 1 ב' - קו תקשורת I^2C מפורט

ניתן לראות שעל שני הקווים SDA (קו הנתון) ו SCL (קו השעון) יכולים להתחבר מספר רכיבים . לכל רכיב יש כתובת ייחודית משלו.

באיור רואים 2 רכיבים המתחברים על הקווים. בחלק התחתון של האיור רואים מבנה פנימי של רכיב ורואים שהרכיב מתחבר בעזרת חוצץ (מתואר על ידי המשולש) המקבל נתון מהקו. מעל החוצץ יש טרנזיסטור בחיבור קולט פתוח(Open Collector) , או טרנזיסטור תופעת שדה - FET בחיבור מרזב פתוח (Open Drain), שיכול לכתוב לקו נתון.

לטרנזיסטור יש לחבר נגד חיצוני (Pull – up resistor) שערכו נע בין $2K[\Omega]\sim10K[\Omega]$. הערכים נבחרים כך שמצד אחד הנגדים לא יהיו קטנים מדי כדי שלא יזרום זרם גדול דרך הקווים ודרך הרכיב (במצב שהרכיב מוציא 0) ומצד שני שהנגד לא יהיה גדול מדי כי הוא קובע את זמן הטעינה והפריקה במעברים בין 0 ל 1 ולהפך ונגד גדול מדי יגביל את קצב התקשורת.

כללים והגדרות בתקשורת I^2C

העברה יכולה להתחיל רק כאשר הקו לא עסוק - NOT BUSY .

בזמן העברת נתון, קו הנתון חייב להישאר יציב כאשר קו השעון במצב גבוה. שינוי בקו הנתון כאשר קו השעון הוא גבוה יתפרש כאותות בקרה.

מגדירים את מצבי הפס הבאים :

Bus Not Busy - פס לא עסוק

גם קו הנתון וגם קו השעון בגבוה.

START DATA TRANSFER - התחל העברת נתון

שינוי במצב קו הנתון מגבוה לנמוך כאשר השעון נמצא בגבוה מוגדר כמצב START .

STOP DATA TRANSFER - עצור העברת נתון

שינוי במצב קו הנתון מנמוך לגבוה כאשר השעון במצב גבוה מוגדר כמצב STOP .

DATA VALID - תקפות נתון

מצב קו הנתון מייצג תקפות הנתון כאשר לאחר מצב START, קו הנתון יציב למשך הזמן הגבוה של אות השעון. הנתון בקו חייב להשתנות רק בזמן מצב נמוך של אות השעון. יש פולס שעון אחד עבור כל ביט של נתון.

כל העברת נתון מתחילה עם מצב START ומסתיימת עם מצב STOP. כמות הבתים המועברת בין START ל STOP לא מוגבלת ונקבעת על ידי רכיב ה MASTER. האינפורמציה מועברת בית אחר בית וכל מקלט מאשר קבלת הבית עם ביט תשיעי של ACKNOWLEDGE .

ב START con כל 8 הביטים מוזזים לרגיסטר SSPSR, ונדגמים בעליה של שעון. הערכים של רגיסטר SSPSR מושווים עם הערכים של רגיסטר SSPADD. הכתובת מושווית בירידה השמינית של השעון. אם ישנה התאמה, הביטים של BF ו SSPOV מאופסים אזי :

הערך של רגיסטר SSPSR נטען לרגיסטר SSPBUF.

bit (Buffer Full) BF עולה ל 1 לוגי.

מופעל פולס $\overline{ACK}$.

ביט דגל הפסיקה של MSSP, SSPIF (PIR1 < 3 >) , עולה ל 1 לוגי בירידה של הפולס התשיעי של השעון.

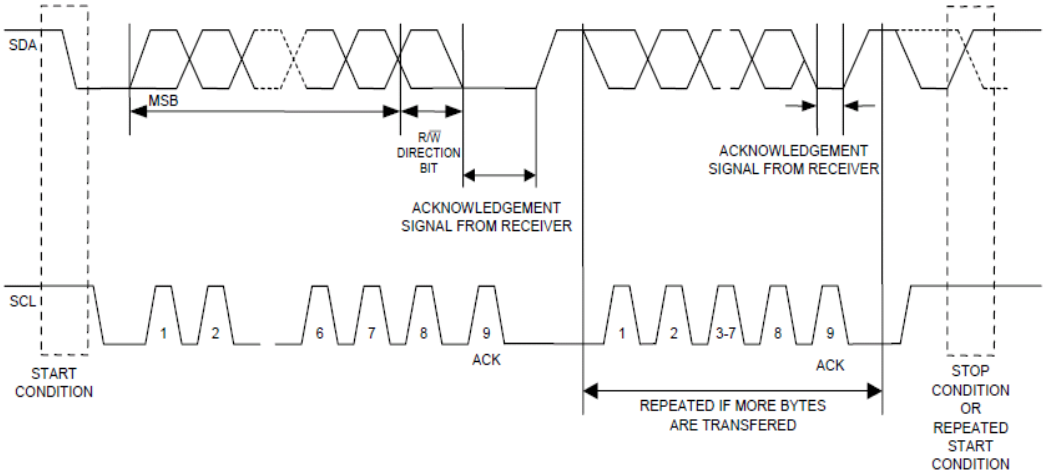
הערה : בהגדרות של I^2C יש תקן של קצב ב $100K[Hz]$ ויש תקן ל 400K[Hz] .

ACKNOWLEDGE – אישור

כל רכיב קולט חייב בסיום קליטת בית שהועבר אליו ליצור ביט ACKNOWLEDGE. רכיב ה MASTER יוצר פולס שעון נוסף הקשור לביט זה.

רכיב היוצר ACKNOWLEDGE חייב להוריד את קו הנתון הטורי - SDA ל 0 בזמן פולס השעון, כלומר שקו הנתון יהיה יציב בנמוך בזמן שקו השעון בגבוה. רכיב ה MASTER מסמן ל SLAVE על סיום התקשורת על ידי **אי יצירת** ביט ה ACKNOWLEDGE כאשר הוא קלט את הביט האחרון מה SLAVE. במקרה כזה על ה SLAVE להשאיר את קו הנתון בגבוה כדי לאפשר ל MASTER ליצור מצב STOP .

באיור 2 ניתן לראות העברה של נתון טורי.



איור 2 - העברת נתון בקו תקשורת טורית I^2C

את הקו SCL (הקו התחתון בשרטוט) יוצר תמיד ה MASTER. יש לשים לב שמצב START קורה כאשר קו SCL בגבוה ואז ה MASTER מוריד את קו הנתון ל 0. לאחר מכן ה MASTER יוצר 8 פולסי שעון ואז הוא שולח בקו הנתון SDA 8 ביטים. 7 ביטים הם כתובת הרכיב והביט ה 8 אומר האם הוא רוצה לכתוב אל הרכיב או לקרא ממנו (0 - כתיבה, 1 - קריאה). לאחר מכן ה MASTER יוצר פולס 9 נוסף שבו ה SLAVE צריך להחזיר ACKNOWLEDGE. לאחר מכן אין צורך ב START נוסף והביטים נשלחים אחד אחרי השני כאשר הצד הקולט נותן ACKNOWLEDGE בביט ה 9. מצב STOP (או START חוזר) מתואר בצד ימין של איור 6. הוא נוצר כאשר קו השעון ב 1 ואז בקו הנתון יש מעבר מ 0 ל 1. מצב START חוזר משורטט בקו מקווקו ובו רואים שבזמן שקו השעון ב 1 יורד קו הנתון ל 0.

העברת נתון בתקשורת I^2C

שתי אפשרויות העברת נתונים קיימות בקו תקשורת I^2C :

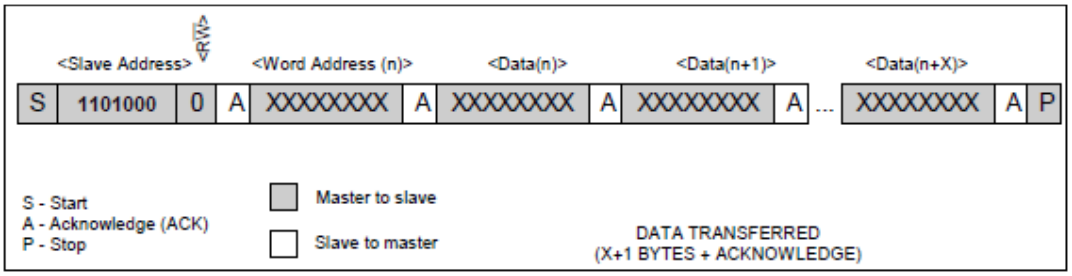
א. ה MASTER משדר וה SLAVE קולט - אופן כתיבה - Write Mode

במקרה זה הבית הראשון המשודר על ידי ה MASTER הוא הכתובת של ה SLAVE. לאחר מכן יבואו מספר ביטים של נתונים. ה SLAVE מחזיר ACKNOWLEDGE בסיום כל בית נתונים שקלט. הנתון מועבר עם ביט ה MSB הראשון !!

ב. ה SLAVE משדר וה MASTER קולט - אופן קריאה - Read Mode

במקרה זה הביט הראשון שנשלח הוא על ידי ה MASTER השולח את כתובת ה SLAVE שמחזיר את ביט ה ACKNOWLEDGE. מכאן ה SLAVE שולח מספר ביטי נתונים. ה MASTER מחזיר ביט ACKNOWLEDGE אחרי כל קליטת בית חוץ מהבית האחרון שהוא איננו מחזיר ACKNOWLEDGE או אפשר להגיד שהוא מחזיר Not ACKNOWLEDGE .

אופן כתיבה - ה MASTER משדר אל אחד מה SLAVES



איור 3 - אופן כתיבת נתון מה MASTER כשה SLAVE הוא המקלט .

באיור 3 מתואר מצב שבו ה MASTER כותב אל ה SLAVE המשמש כמקלט. החלק הכהה שבאיור הוא מה ששולח ה MASTER. החלק הבהיר הוא מה ששולח המקלט - ה SLAVE .

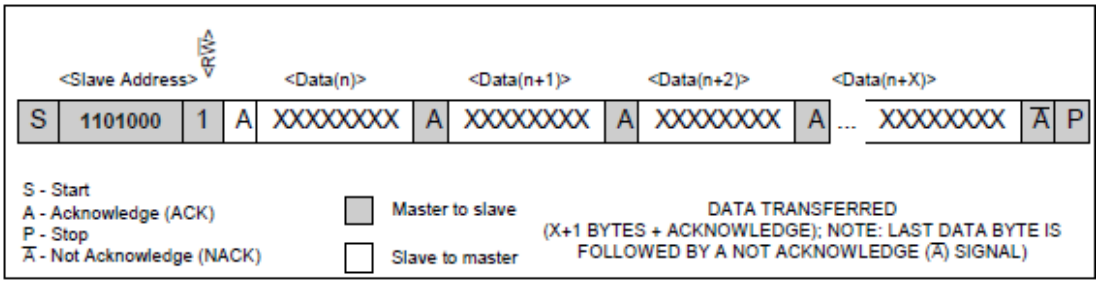
ה MASTER יוצר מצב START (מסומן ב S). לאחר מכן הוא שולח 7 ביטים של כתובת הרכיב והביט ה 8 הוא 0 המציין שהוא הכותב וה SLAVE הוא המקלט. על ה SLAVE לענות ב ACKNOWLEDGE (מסומן ב A). לאחר מכן ה MASTER שולח בית נוסף הטוען את מצביע (אוגר) הכתובות בתוך הרכיב. הנתון הבא נכתב לכתובת זו ומצביע הכתובות גדל אוטומטית ב 1. כל נתון נכתב בכתובת שבמצביע הכתובות, ומצביע הכתובות מתקדם ב 1. אחרי כל בית שנקלט על ידי ה SLAVE הוא שולח אישור שקלט - ACKNOWLEDGE . ה MASTER מסיים את התקשורת בעזרת מצב STOP (מופיע בצד ימין עם האות P) .

אופן קריאה - ה SLAVE משדר אל ה MASTER

גם מצב זה מתחיל תמיד במצב שבו ה MASTER משדר אל ה SLAVE אבל כאן הוא אומר שהוא רוצה לקרוא ממנו. הביט הראשון שה MASTER משדר נקלט על ידי ה SLAVE כמו שתואר בפסקה הקודמת, כלומר ה MASTER יוצר מצב START, שולח 7 ביטים של כתובת ה SLAVE אבל הביט השמיני יהיה 1 שבו הוא אומר שהוא רוצה לקרוא. מכאן ה SLAVE משדר את הנתונים וה MASTER עונה עם ביט ACKNOWLEDGE. בבית האחרון, כשה MASTER רוצה לסיים את התקשורת , הוא איננו מגיב בביט ה 9 ולא שולח ACKNOWLEDGE (מסומן באיור ב $\bar{\mathbf{A}}$) וגם שולח ביט עשירי של STOP .

הנתונים המשודרים מה SLAVE מתחילים מהכתובת האחרונה שבה נמצא מצביע הכתובות. כל נתון שה SLAVE שולח הוא מקדם את מצביע הכתובות לכתובת הבאה (אוטומטית) .

איור 4 מתאר מצב תקשורת זה. גם כאן הצבע הכהה הוא של ה MASTER והבהיר של ה SLAVE .

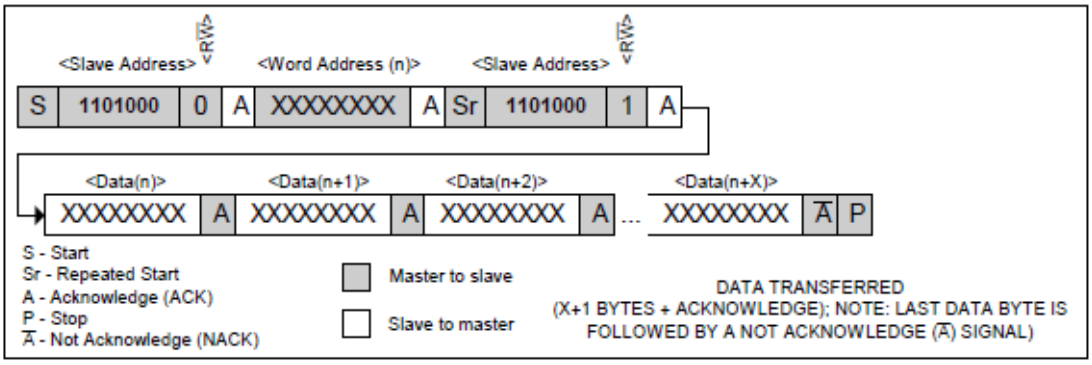


איור 4 - אופן קריאה - ה MASTER הוא המקלט

בדרך כלל תהליך התקשורת יהיה הבא : ה MASTER יכתוב 2 בתים אל הרכיב. בבית הראשון הוא אומר לרכיב שהוא פונה אליו לכתיבה (בנוסף לכתובת של ה SLAVE). בבית השני הוא טוען את מצביע (אוגר) הכתובות בתוך הרכיב. מיד לאחר מכן יישלח STOP (או START חוזר - Sr) ואז יבצע תקשורת חדשה שבה הוא יפנה לרכיב לקריאה מהכתובת ששלח אליו בפעולת הכתיבה. ה SLAVE שלוח ביט ACKNOWLEDGE, ואז ה MASTER מתחיל לשלוח בתים של נתונים שנכתבים לכתובת של מצביע הכתובות שגל אוטומטית ב 1 לאחר כל העברת בית.

בבית האחרון, ה MASTER שולח $\overline{ACK}$ ($\bar{A}$ – כלומר לא שולח ACKNOWLEDGE), ומיד לאחר מכן STOP .

איור 5 מתאר פעולת כתיבה וקריאה מהכתובת הרצויה. גם כאן הצבע הכהה הוא של ה MASTER והבהיר של ה SLAVE .



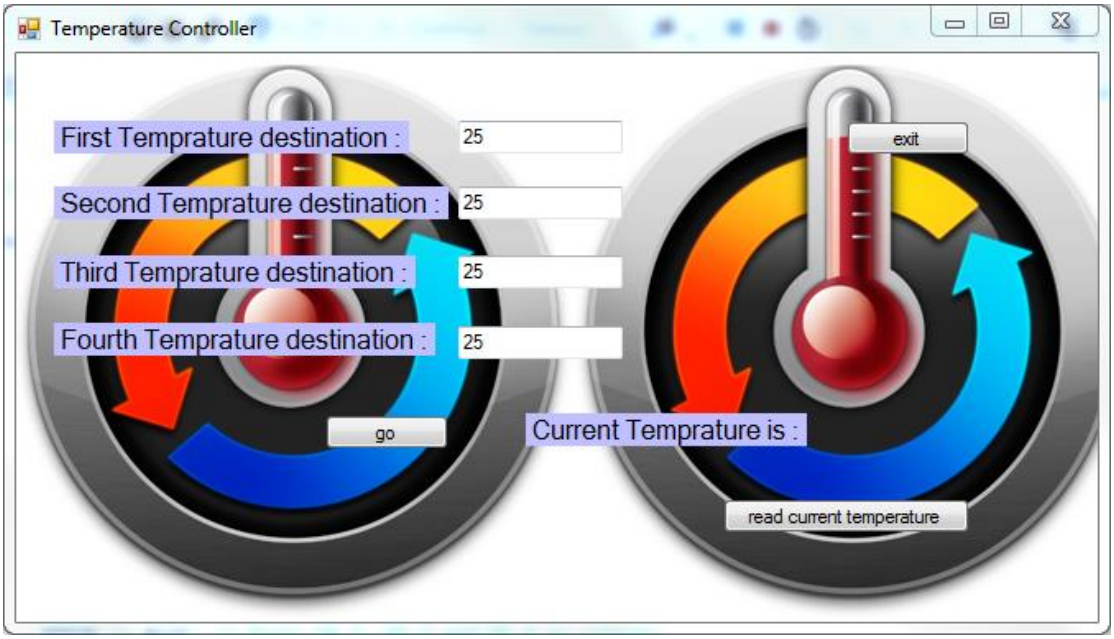
איור 5 - פעולה משולבת של כתיבה ל SLAVE כדי לציין כתובת רצויה וקריאה מה SLAVE

6. שימוש בתקשורת I2C

באמצעות תקשורת זו בקר ה MSP מתקשר עם בקר הטמפרטורה. השליטה על המאוורר, על נגד ההספק, וקריאת הטמפרטורה הנוכחית מחיישן הטמפרטורה מתבצעות בעזרת פרוטוקול תקשורת זה.

תקשורת טורית זו הינה סינכרונית ובה מועברים פולסי שעון מה Master (בקר ה MSP) אל ה Slave (חיישן הטמפרטורה).

7. פאנל המשתמש



כאן מכניס המשתמש ארבע טמפרטורות רצויות. הפעלת התוכנית מתאפשרת בלחיצה על לחצן "go".
בכל זמן נתון ניתן ע"י לחיצה על לחצן "read current temperature" להציג על גבי מסך זה את הטמפרטורה העדכנית הנמדדת ע"י חיישן הטמפרטורה.

8. בקרת PID

לשם קבלת החלטה לשליטה על המאוורר ועל נגד ההספק, בוצעה בתוכנית בקרה על הטמפרטורה המבוססת PID.

בקר ה PID – Proportional Integral Derivative הינו הנפוץ ביותר בבקרת מערכות לינאריות.

P : פרופורציונלי. מתקן את השגיאה העכשווית בצורה פרופורציונלית לשגיאה.

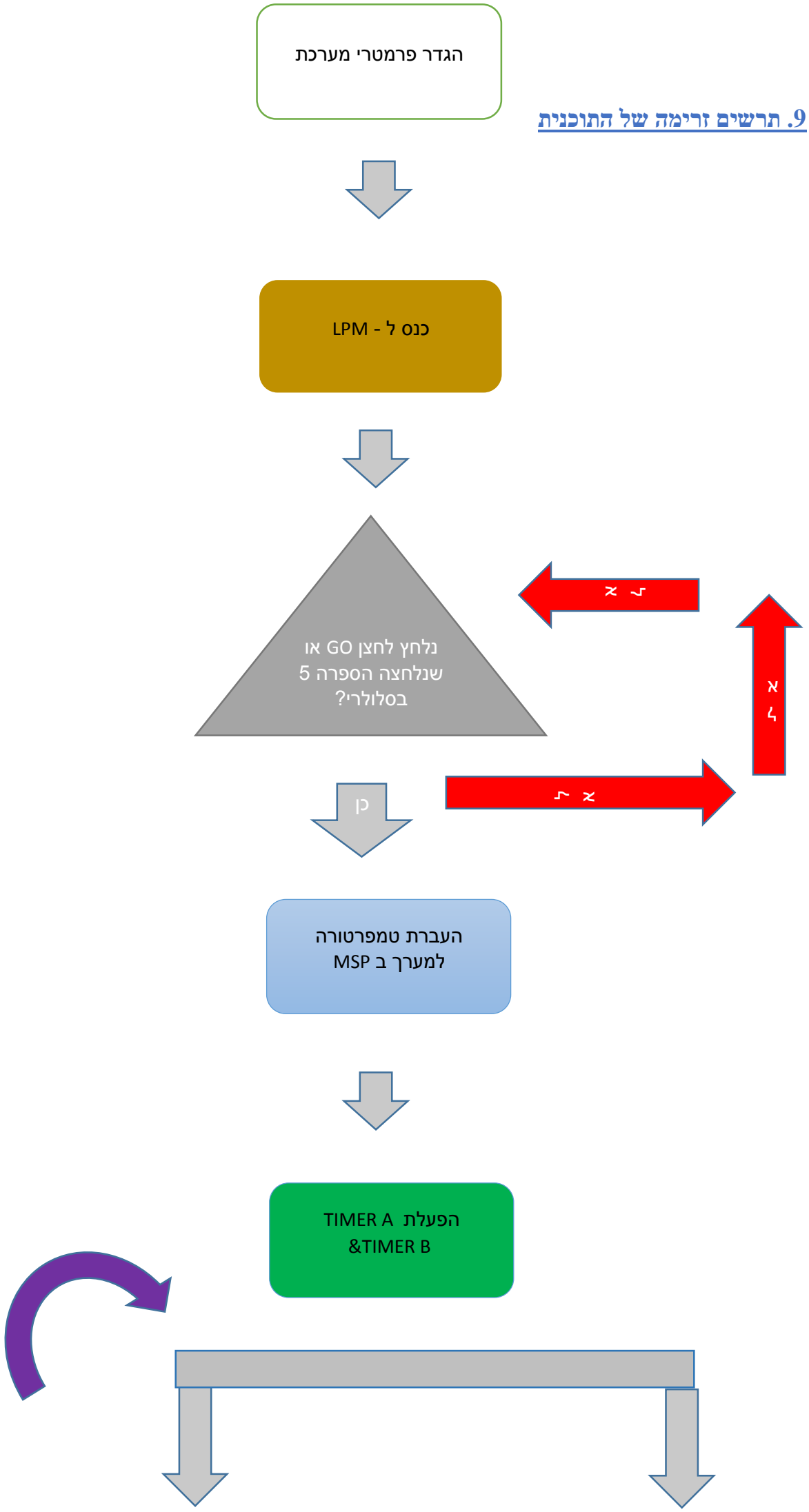
I : אינטגרלי. מוסיף תיקון פרופורציונלי לאינטגרל בזמן על שגיאת העקיבה ובכך מבטיח שגיאת מצב מתמיד אפסית.

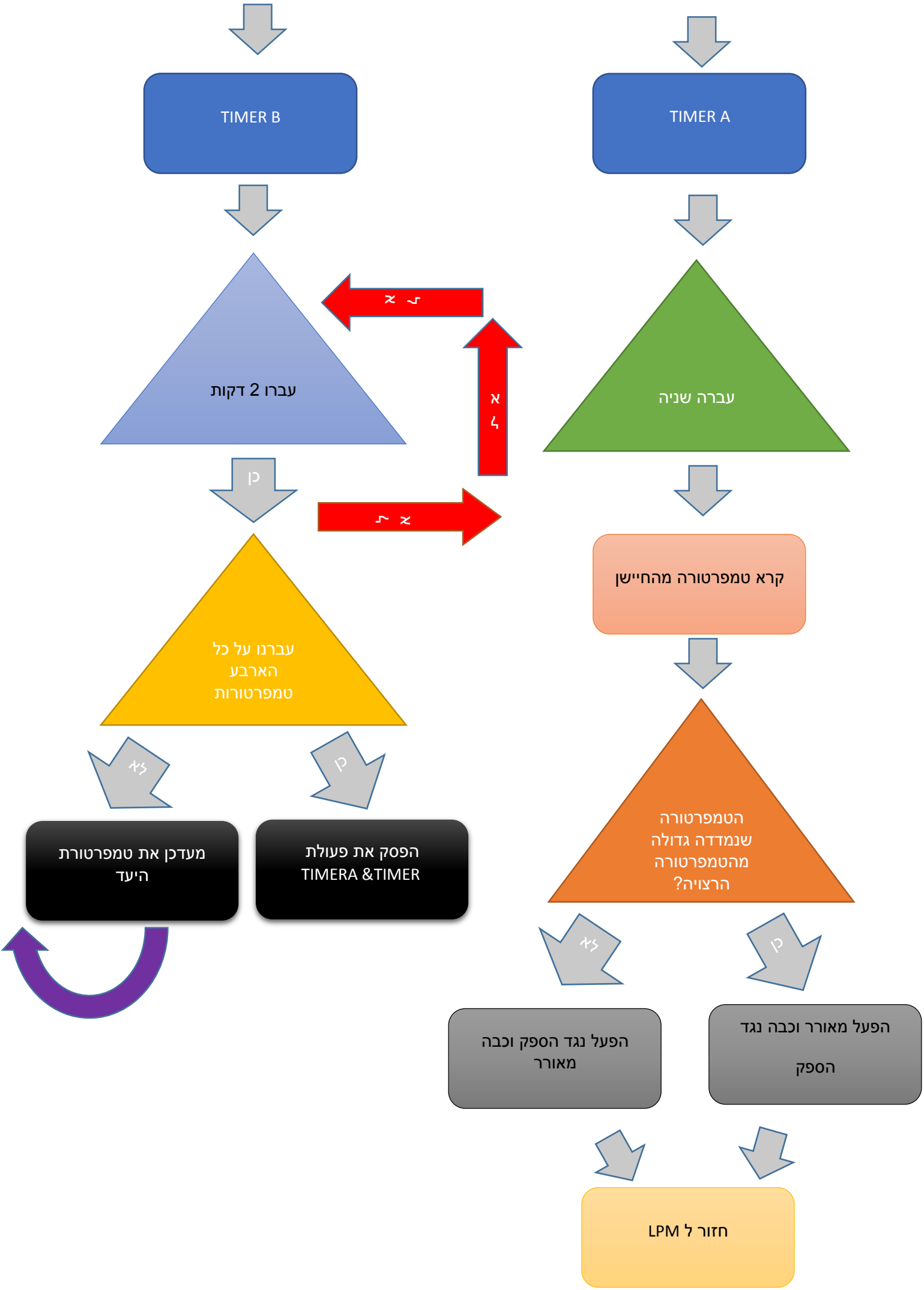
D : דיפרנציאלי. מוסיף תיקון פרופורציונלי לנגזרת בזמן של השגיאה ובכך מוסיף ריסון למערכת ומונע תגובת יתר (Overshoot).

בתוכנית, כל עוד השגיאה הכוללת מחוץ "לשרוול" שסביב טמפרטורת היעד, המשך להפעיל את מה שפועל (מאוורר/נגד הספק) ולשמור על מה שכבוי (מאוורר/נגד הספק).

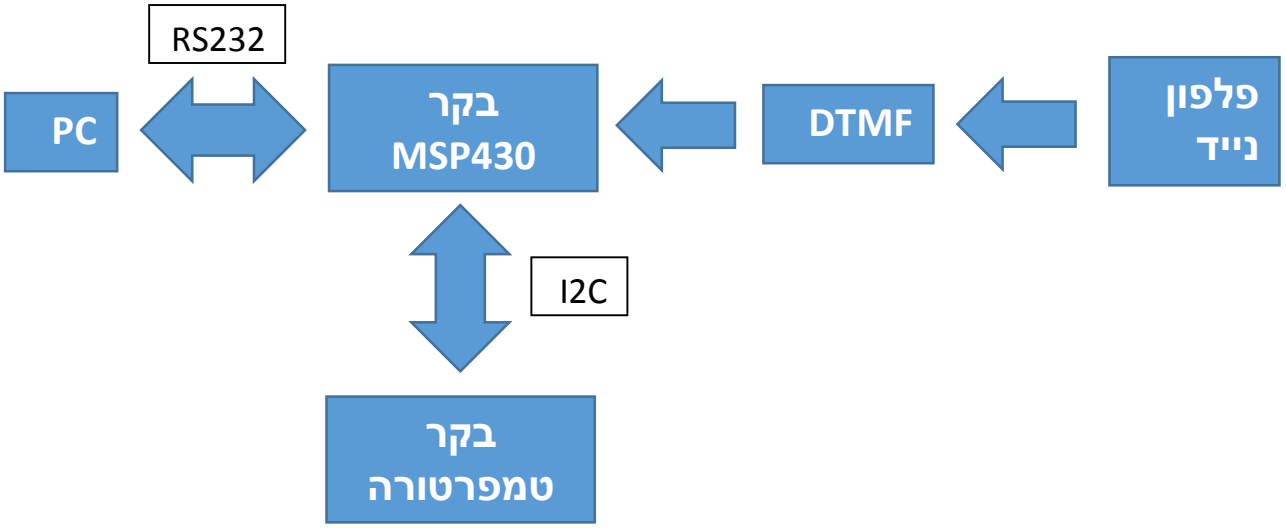
חישוב השגיאה הכוללת נעשה בעזרת חישוב כל אחד ממרכיבי ה PID ומכפלתם בהגבר (Gain) מתאים.

9. תרשים זרימה של התוכנית




TIMER B
TIMER A
עברו 2 דקות
עברה שניה
לא
לא
לא
לא
כן
קרא טמפרטורה מהחיישן
עברנו על כל
הארבע
טמפרטורות
הטמפרטורה
שנמדדה גדולה
מהטמפרטורה
הרצויה?
לא
כן
לא
כן
מעדכן את טמפרטורת
היעד
הפסק את פעולת
TIMERA &TIMER
הפעל נגד הספק וכבה
מאורר
הפעל מאורר וכבה נגד
הספק
חזור ל LPM

10. דיאגרמת בלוקים של המערכת



11. קוד התוכנית

11.1. קוד ה IAR

```
//Student Names : Amir Mallak      203764683
//                Moshe Bensemon   300779022
//Project Name : A temperature PID controller.
//Note : The signals could be sent either from the PC or from the phone.
//       If it is sent from the phone, the tones are transmitted to signals
using DTMF reciever.


#include <msp430xG46x.h>

#define TMP102_i2c_addr 0x90


//*************************************
//         Global Variables
//*************************************

int twoMinuts = 0; //Count 2 minuts

double PIDError; //The total final error of the PID controller

double err_old; //Old error (the error in the previous step)

double err = 0; //The current error (the difference between the destination
temperature and the measured temperature)

double P_err; //The Proportional error = the current error

double I_err = 0; //The Integral error = the sum of the old errors untill now

double D_err; //The Derivative error = the difference between the current error
and the old error

int destinationTempIndex = 0; //the index of the 4 tempratures array (wanted
temp)

double destinationTemp[4] = {25,25,25,25}; //4 wanted temps- updated by user

double DTMFPress[8] = {0};  //save 8 presses of the phone

unsigned int currentPhonePress = 0; //save current DTMF call

double measuredTemp = 0; //contains the converted temprature

double MSBTemp, LSBTemp; //Represents the two chars (respectively) which were
sent from the PC or from the phone (each char represents one digit of the
destination temperature)

unsigned char getTemp_PC[8] = {0}; //An array to save the chars that came from
the PC

double currentDestinationTemp; //The current temperature which we want to reach

unsigned int i, j = 0;

char receivedTemp[2] = {0}; //array that contains tempratures came from i2c
```

```
//*********************************************************
//                      I2C
//*********************************************************

void wait(long time)
{
       for( ; time > 0; time -- );
}


void i2c_init()
{
       //SDA = P3.1, SCL = P3.2

       P3DIR |= 0x06;       // Output (sets P3.1(SDA) and P3.2(SCL) as
outputs("1"))

       P3OUT &= ~0x06; // initial zero (SDA and SCL are down "0") on data bus
(so we can now the "values" on the bus, not garbage)

       wait(10);

       P3OUT |= 0x04; // SCL up (P3.2). we pull the SCL up because the SDA
could only change if the SCL are "1", otherwise it will be considered as a

       wait(50);

       P3OUT |= 0x02; // SDA up (P3.1). SCL "1", SDA from "0" to "1" -> STOP.
                      //Now because both SDA and SCL are "1", this means "Bus
Not Busy"

       P3SEL |= 0x06; // P3.1 + P3.2 are set as peripherials

       //******************** i2c configurations*********************

       UCB0CTL1 |= UCSWRST; // Page 638. Enable software reset

       UCB0CTL0 = UCMST + UCMODE_3 + UCSYNC; // Master mode, I2C mode,
synchronous mode

       UCB0CTL1 = UCSSEL_2 + UCSWRST; // Use SMCLK (for peripherial ports),
keep software reset enabled

       UCB0BR0 = 11; // SMCLK/11 = 95.3kHz (I2C has two modes, one 400K[Hz] and
another 100K[Hz])

       UCB0BR1 = 0x00;

       UCB0I2CSA = 0x48; // Setting the slave Address to 48H

       UCB0CTL1 &= ~UCSWRST; // Clear software reset, resume operation
```

```
	IE2 |= UCB0TXIE + UCB0RXIE; //USCI_B0 transmit interrupt enable, USCI_B0
receive interrupt enable
}

void i2c_recv(char address, int twoBytes, char *result)
{
	while (UCB0STAT & (UCSCLLOW | UCBBUSY) ) ; // Page 640. communication
problem due to SCL is held low or Bus "Busy"
								// The I2C communication
cannot beggin untill the line is "Not Busy"

	UCB0I2CSA = address>>1; // Page 625, I2C Master Transmitter Mode, line
1+2. shift right by 1 position so that the adderss will be right justified ->
the address is changed to 0X48

	UCB0CTL1 |= UCTR + UCTXSTT; // Page 638. I2C generate start condition,
transmitter

	UCB0TXBUF = 0xEE; // start conversion

	while(!(IFG2&UCB0TXIFG)); // Wait untill the data is transmitted

	wait(20000);

	UCB0CTL1 |= UCTR + UCTXSTT; // I2C start condition

	UCB0TXBUF = 0xAA; // read temprature

	while(!(IFG2&UCB0TXIFG)); // Wait for finish of transmitting data


	IFG2 &= ~UCB0RXIFG; // reset receive flag

	UCB0CTL1 &= ~UCTR; // I2C receiver mode (when the UCTR is set then we
are in transmitter mode, and when it is not set then we are in receiver mode)

	UCB0CTL1 |= UCTXSTT; // I2C start condition, receiver (because we
haven't set the UCTR)

	while(twoBytes-- > 0) { // Receive 2 bytes
		while (!(IFG2 & UCB0RXIFG)); //waits untill Rx buffer is full.
this flag indicates that a byte is waiting in the buffer
		*(result++) = UCB0RXBUF;
	}

	UCB0CTL1 |= UCTXSTP; // Page 638. I2C generate stop condition

	while(UCB0CTL1 & UCTXSTP); // check that the stop condition is through
(if the UCTXSTP bit has went down in UCB0CTL1)

	wait(10);

	i2c_init(); // re-set the i2c module

}
//************************************************************
//************************************************************
//************************************************************
```

```
//*****************************************************************
// RECEIVE INTERRUPT FUNCTION RS232
//*****************************************************************

#pragma vector=USCIAB0RX_VECTOR

__interrupt void USCIA0RX_ISR (void)
{
	TBCCTL0 = ~CCIE; // TBCCR0 interrupt disabled
	TACCTL0 = ~CCIE; // TACCR0 interrupt disabled
	destinationTempIndex = 0;
	twoMinuts = 0;

	while(!(IFG2&UCA0RXIFG)); //waits untill Rx buffer is full
	getTemp_PC[i]=UCA0RXBUF; //receives all 8 characters of the wanted
temprature
	i++;

	if(i == 8)
	{
		j = 0;
		for (i = 0; i <= 7; i = i+2)
		{
			MSBTemp = getTemp_PC[i]-'0';
			MSBTemp *= 10;
			LSBTemp = getTemp_PC[i+1]-'0';
			destinationTemp[j] = MSBTemp + LSBTemp;
			j++;
		}
		i=0;

		currentDestinationTemp = destinationTemp[destinationTempIndex];
		destinationTempIndex ++;
		TBCCTL0 = CCIE; // TBCCR0 interrupt enabled
		TACCTL0 = CCIE; // TACCR0 interrupt enabled
	}
}




//*************************************************************************
***
//                      PID Controller (Proportional-Integral-Derivative)
//*************************************************************************
***

// PID = GainP * actual error + GainI * SUM(previous errors) + GainD * (actual
error - last error)
// error = sp(set point) - pv(process value)
// sp : currentDestinationTemp, pv : measuredTemp

double PID (double sp, double pv)
{
  err_old = err;
  err = sp - pv;

  P_err = err;
  I_err += err_old;
```

```
  D_err = err - err_old;

  return 0.1*P_err + 0.3*I_err + 0.02*D_err;
}



//*******************************************************************
//                           Timer_A ISR - counts 1 second
//*******************************************************************

#pragma vector=TIMERA0_VECTOR

__interrupt void Timer_A (void)
{
       i2c_recv(TMP102_i2c_addr,2,receivedTemp); //Reads the temperature into
"receivedTemp[0]" and "receivedTemp[1]"


       while(!(IFG2&UCA0TXIFG));   // wait for UCA0TXBUF to finish transmitting
to the pc
       UCA0TXBUF = receivedTemp[0]; //The transmitter's buffer gets the
temperature which came from the I2C (to transmit it to the PC in case the
button "read current temperature was pressed")


       measuredTemp = 16*receivedTemp[0] + receivedTemp[1]/16;
       measuredTemp = measuredTemp * 0.0625;

       PIDError = PID(currentDestinationTemp, measuredTemp);
    PIDError = PIDError > 0 ? PIDError : -PIDError;

       if((err < 0) && (PIDError > 0.05*currentDestinationTemp)) //cooler
       {
              P3OUT |=  0x20; // turn off the heater P3.5
              P3OUT &= ~0x10; // turn on the fan P3.4
       }

       else if ((err > 0) && (PIDError > 0.05*currentDestinationTemp)) //heater
       {
              P3OUT |= 0x10;// turn off the fan P3.4
              P3OUT &= ~0x20;// turn on the heater P3.5
       }

       __bic_SR_register_on_exit(LPM0_bits); // Exit LPM0
}





//*************************************************************************
******
//                        Timer_B ISR - counts every second to get to 2 minuts
//*************************************************************************
******

#pragma vector=TIMERB0_VECTOR

__interrupt void Timer_B (void)
{
```

```
	if (twoMinuts == 120) //If two minutes have been passed
	{
		if (destinationTempIndex == 4) //If we have already reached all
four temperatures -> "stop" (timer A and timer B are disabled) program
		{
			TBCCTL0 = ~CCIE; // TBCCR0 interrupt disabled
			TACCTL0 = ~CCIE; // TACCR0 interrupt disabled
		}
		else //If we haven't reached all four temperatures yet, then
start counting again and take the next destination temperature
		{
			twoMinuts = 0;
			currentDestinationTemp =
destinationTemp[destinationTempIndex];
			destinationTempIndex ++;
		}
	}

	else //If two minuts haven't been passed yet
	{
		twoMinuts ++;
	}
}



//*******************************************************************
// CONVERT_DTMF Function
//*******************************************************************

// This Function Converts the DTMF Output into Integer

int CONVERT_DTMF(void)
{
	currentPhonePress = P7IN;		// Reading Port 7
	currentPhonePress &= 0x0f;		// Working Only With the 4 LSB (DTMF)

	switch (currentPhonePress)
	{
		case 1: return 1;
		case 2: return 2;
		case 3: return 3;
		case 4: return 4;
		case 5: return 5;
		case 6: return 6;
		case 7: return 7;
		case 8: return 8;
		case 9: return 9;
		case 10: return 0;
	}
	return -1;  // error
}



//*******************************************************************
// GET_DTMF Function
//*******************************************************************

// a function to read the DTMF output
```

```
void GET_DTMF(void)
{
	while((P2IN & 0x80)==0x80);  //while there is an interrupt (the 8th bit
in port 2, is the interrupt bit)
	while(!((P2IN & 0x80)==0x80));  // while interrupt stoped and another
interrupt haven't been pressed
	DTMFPress[0]  = CONVERT_DTMF();  // when another interrupt occurred ->
DTMF tone to Integer

	while((P2IN & 0x80)==0x80);
	while(!((P2IN & 0x80)==0x80));
	DTMFPress[1]  = CONVERT_DTMF();

	while((P2IN & 0x80)==0x80);
	while(!((P2IN & 0x80)==0x80));
	DTMFPress[2]  = CONVERT_DTMF();

	while((P2IN & 0x80)==0x80);
	while(!((P2IN & 0x80)==0x80));
	DTMFPress[3]  = CONVERT_DTMF();

	while((P2IN & 0x80)==0x80);
	while(!((P2IN & 0x80)==0x80));
	DTMFPress[4]  = CONVERT_DTMF();

	while((P2IN & 0x80)==0x80);
	while(!((P2IN & 0x80)==0x80));
	DTMFPress[5]  = CONVERT_DTMF();

	while((P2IN & 0x80)==0x80);
	while(!((P2IN & 0x80)==0x80));
	DTMFPress[6]  = CONVERT_DTMF();

	while((P2IN & 0x80)==0x80);
	while(!((P2IN & 0x80)==0x80));
	DTMFPress[7]  = CONVERT_DTMF();

	j = 0;
	for (i = 0; i <= 7; i = i+2) //Merging each two integers into one
temprature -> converts 8 integers into 4 numbers
	{
		MSBTemp = DTMFPress[i];
		MSBTemp *= 10;
		LSBTemp = DTMFPress[i+1];
		destinationTemp[j] = MSBTemp + LSBTemp;
		j++;
	}
	i=0;

	currentDestinationTemp = destinationTemp[destinationTempIndex]; //
Updates the current temprature. every two minutes(which timer B counts it) the
current temprature is updated agian (the next temperature is taken)
	destinationTempIndex ++;

	TBCCTL0 = CCIE; // TBCCR0 interrupt enabled. Timer_B Capture/Compare
Register 0 interrupt enabled
	TACCTL0 = CCIE; // TACCR0 interrupt enabled. Timer_A Capture/Compare
Register 0 interrupt enabled

}
```

```
//*****************************************************************
// Mask Receive Call Function
//*****************************************************************

// This Function Mask or UnMask the DTMF Interrupt

void MASK_RECIVE_CALL_INTERRUPT(int mask_unmask) //1 = mask the inturrput
{                                                //0 = unmask the interrupt
       if (mask_unmask == 1)
       {
              P2IE &= 0x7f;        // Mask DTMF Interrupt (set the 8th bit to
0)
       }

       if (mask_unmask == 0)
       {
              P2IE |= 0x80;       // UnMask DTMF Interrupt (set the 8th bit to
1)
       }
}




//*****************************************************************
//                  Port2 Interrupt Service Routine
//*****************************************************************
#pragma vector=PORT2_VECTOR
__interrupt void PORT2_ISR (void)
{
       _BIC_SR(GIE);
       int DTMFBeggin;

       ////////////////////////////DTMF interrupt//////////////////////////

       if (P2IFG & 0x80)                          // Receive Call Interrupt
(DTMF)
       {
              DTMFBeggin = (P7IN & 0x0f);                    // Get the 4 LSB

              MASK_RECIVE_CALL_INTERRUPT(1);        // Disable other interrups
until this one is decoded

              if(DTMFBeggin == 0x05)      // if the number "5" were pressed,
then beggin
              {
                     TBCCTL0 = ~CCIE; // TBCCR0 interrupt disabled
                     TACCTL0 = ~CCIE; // TACCR0 interrupt disabled
                     twoMinuts = 0;
                     destinationTempIndex = 0;
                     GET_DTMF();                        // get the 4 numbers
from the phone
              }

              P2IFG &= ~0x80;
              _BIC_SR(GIE);
              P2IFG = 0x00;
              P2IE |= 0x80;
```

```
            MASK_RECIVE_CALL_INTERRUPT(0);            // Enable other
interrups
        }
        else
        {
            P2IFG = 0x00 ; // if noise (not interrupt)
        }

        _BIS_SR(GIE);
        return;
}



//*****************************************************************

//                          Main routine

//******************************************************************

void main (void)
{
        WDTCTL = WDTPW + WDTHOLD; // Stop WDT


        // Port 3 configuration: P3.4 = Fan_Con; P3.5 = Hot_Con;

        P3DIR |= 0x30;  // P3.5, P3.4 - OUTPUT(Hot con, Fan con)
        P3DIR &= ~0x01;  // P3.0 == INPUT (TOUT)
        P3OUT |= 0x30; // OUTPUT PINS ARE 1 (FAN CON & HOT CON)
        P3SEL &= ~0x31; // set pins as I/O

        ////////////////////////////////////////////////////


        P5DIR |= 0x1E; // Ports P5.2, P5.3 and P5.4 as outputs
        P5SEL |= 0x1E; // Ports P5.2, P5.3 and P5.4 as special function (COM1,
COM2 and COM3)



        // DTMF Ports Configuration

        P2DIR = 0x00;               // P2.7-DTMF Interrupt input
        P2SEL = 0x00;               // Set the Port as I/O
        P2IES = 0x00;                // Low to high
        P2IFG = 0x00;               // Interrupts flags down
        P2IE  |= 0xFF;                      // Interrupt Enable port 2.7 - DTMF

        // DTMF inputs

        P7DIR &= ~0x8f;              // 7.0,7.1,7.2,7.3 DTMF Input
        P7SEL &= ~0x8f;              // 7.0,7.1,7.2,7.3 I/O



        //-----------------------------------initialization----------------------
--------------

        i2c_init();
```

```
        ////////////////////////// RS232 Comm Configuration
//////////////////////////////////

        P2SEL |= 0x030; // P2.5,4 = USCI_A0 RXD/TXD

        UCA0CTL1 |= UCSSEL_1;  // CLK = ACLK


        UCA0BR0 = 0x03; // 32k/9600 - 3.41

        UCA0BR1 = 0x00; // 32K

        UCA0MCTL = 0x06; // Modulation page 572 Table 19-4

        UCA0CTL1 &= ~UCSWRST; // **Initialize USCI state machine**

        IE2 |= UCA0RXIE; // Enable USCI_A0 RX interrupt*/

        ////////////////////////////////////////////////////////////////////
/

        //------------------------------------timer A configuration-------------
----------------

        TACCR0 = 0X7D00;  //0X7D00 equals to 32K, so time*frequency = count =>
time = count/frequency = 32K/32K = 1 sec

        TACTL = TASSEL_1 + MC_1; // SMCLK, up mode

        //--------------------------------------------------------------------
----------------


        //------------------------------------timerB b configuration------------
----------------

        TBCCR0 = 0X7D00; // counts to 32k, i.e 1 sec

        TBCTL = TBSSEL_1 + MC_1; // SMCLK, up mode

        //--------------------------------------------------------------------
----------------

        _BIS_SR(LPM0_bits + GIE); // Enter LPM0, interrupts enabled

        while(1);
}
```

11.2. קוד ה #C

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;

namespace finel_proj_DCS
{

    public partial class read_current_temp : Form
    {
        ////// Default Parameters //////
        int baudRate = 9600;
        int parityType = 0;
        int wordWidth = 8;
        int stopBits = 1;
        string port = "COM1";
        /// ////////////////////////////
        int i = 0;
        string toStr;
        double currentTemp = 0;
        decimal decValue; // convert the char received from the MSP to decimal
value

        public read_current_temp()
        {
            InitializeComponent();
            ////////////////////////////////// serialPort configurations
//////////////////////////////////

            serialPort.BaudRate = baudRate;
            serialPort.DataBits = wordWidth;
            serialPort.PortName = port;

            switch (stopBits)
            {
                case 1:
                    serialPort.StopBits = System.IO.Ports.StopBits.One;
                    break;
                case 2:
                    serialPort.StopBits = System.IO.Ports.StopBits.Two;
                    break;
            }


            switch (parityType)
            {
                case 0:
                    serialPort.Parity = System.IO.Ports.Parity.None;
                    break;
                case 1:
                    serialPort.Parity = System.IO.Ports.Parity.Even;
                    break;
                case 2:
                    serialPort.Parity = System.IO.Ports.Parity.Odd;
```

```
                    break;
            }


            ////////////////////////// opening serial port
///////////////////////
            serialPort.Open();

////////////////////////////////////////////////////////////////////


            /////////////////////////// defaul temperature values
//////////////////

            first_temp.Text = "25";
            second_temp.Text = "25";
            third_temp.Text = "25";
            fourth_temp.Text = "25";
        }

        private void go_button_Click(object sender, EventArgs e)
        {

            timer1.Enabled = true;
            timer1.Start();

        }

        private void timer1_Tick(object sender, EventArgs e)
        {
            char[] send = new char[1];

            char[] temperaturs = new char[8];
            if (i == 8) //if all the tempratures were sent
            {
                timer1.Stop();
                i = 0;
            }

            else
            {
                temperaturs[0] = first_temp.Text[0];
                temperaturs[1] = first_temp.Text[1];
                temperaturs[2] = second_temp.Text[0];
                temperaturs[3] = second_temp.Text[1];
                temperaturs[4] = third_temp.Text[0];
                temperaturs[5] = third_temp.Text[1];
                temperaturs[6] = fourth_temp.Text[0];
                temperaturs[7] = fourth_temp.Text[1];


                send[0] = temperaturs[i]; //send 8 chars representing the
tempratures to MSP
                serialPort.Write(send, 0, 1); //Writes a specified number of
characters to the serial port using data from a buffer

                i = i + 1;
            }
        }
```

```
        private void serialPort_DataReceived(object sender,
System.IO.Ports.SerialDataReceivedEventArgs e)    // Enter when data is recived
from serial port
        {
            char[] receive = new char[1];
            serialPort.Read(receive, 0, 1); //Reads a number of characters from
the SerialPort input buffer and writes them into an array of characters at a
given offset. (Buffer, Offset (to read from the buffer), Count(how many chars
to read))
            decValue = receive[0];

            currentTemp = (double)decValue;
            toStr = currentTemp.ToString();  //Returns a string that represents
the current object -> convert the received temperature to string

            serialPort.DiscardInBuffer(); // Discards data from the serial
driver's receive buffer. It clears the receive buffer, but does not affect the
transmit buffer
        }


        private void read_temperature_Click(object sender, EventArgs e)
        {
            label1.Text = "The current temperature is : ";
            currentTempLabel.Text = toStr;  //display the string of the
temperature on the screen
        }



        private void exit_Click(object sender, EventArgs e)
        {
            serialPort.Close();
            Close();
        }

        private void currentTempLabel_Click(object sender, EventArgs e)
        {

        }

        private void read_current_temp_Load(object sender, EventArgs e)
        {

        }

    }
}
```