IOT – תקשורת באמצעות רטט

פרויקט סיום – "קורס מתקדם במערכות מחשב"

נתנאל קצבורג ואיתי חי

בהנחיית פרופ' סיון טולדו

26.3.17

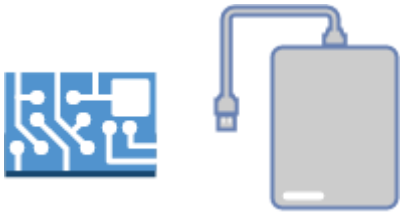
תוכן עניינים

1. מבוא

במסגרת פרויקט הסיום בקורס "קורס מתקדם במערכות מחשב" בחרנו לבנות מערכת המממשת אותנטיקציה בין מיקרו-בקר וחיישן המחובר אליו אל מכשיר טלפון מעל רטט. המיקרו-בקר במערכת, אשר מתפקד כשרת BLE ומייצא מספר שירותים שונים, יסכים לבצע Pairing עם מכשיר חיצוני רק אם תתקבל סיסמה אשר הועברה אליו קודם לכן באמצעות רטט. לאחר האימות, המשתמש יהיה חשוף לנתוני החיישן דרך שירות ייעודי.

<u>חלקי הפרויקט</u>:

א. חיבור חיישן GY-801 ללוח CC2650.



ב. העברת מפתח (PIN code) ממכשיר סלולרי ללוח באמצעות רטט . זהו החלק המרכזי של הפרויקט.

ג. פיתוח מעל BLE – כתיבת GATT Service שישמש כממשק לסטטוס החיישן עבור משתמש חיצוני באפליקציה (קריאה בלבד).

2. חומרה בשימוש

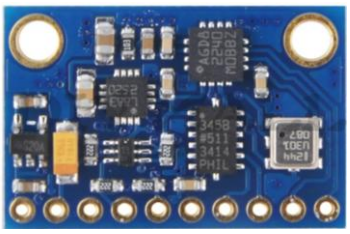
א. מיקרו-בקר – SimpleLink CC2650 Wireless MCU LaunchPad.

ב. יחידת מדידה אינרציאלית (IMU) - GY-801.

ה-IMU היא מערכת אלקטרונית אשר מודדת את הכוחות והמומנטים הפועלים עליה, את השדה המגנטי והטמפרטורה שסביבה, או את השינויים באותם מדדים. המדידה נעשית באמצעות שילוב מדי תאוצה, ג'יירוסקופים ומגנטומטרים. בפרויקט זה אנו מבצעים שימוש רק בקריאות ממד התאוצה ללא החיישנים הנוספים על הלוח, בכדי להשתמש ברטט כערוץ תקשורת.

בחרנו להשתמש ב-IMU שכולל ארבעה חיישנים (10DOF):

- L3G4200D [1] – ג'ירוסקופ הוא מכשיר מדעי המשמש למדידה או שמירה של יציבות ,תוך התבססות על עקרונות שימור התנע הזוויתי. לחיישן הנ"ל יש שלושה צירים שעבורם הוא מודד את השינויים בתנע הזוויתי.
- ADXL345 [2] – מד תאוצה בעל שלושה צירים. מבחין בשדה הכבידה של כדוה"א ובנוסף יודע לזהות שינוי דינמי בתאוצה כתוצאה מתנועה או מכה.
- HMC5883L [3] – מגנטומטר בעל שלושה צירים, שמסוגל להבחין בשינויים של החיישן ביחס לשדה מגנטי בכלל, ושל כדוה"א בפרט.
- BMP180 [4] – חיישן ברומטרי, ניתן להשתמש בו למדידת לחץ או בכדי להעריך גובה ביחס לפני הים.



תרשים 1: חיישן GY-801 על breakboard

3. סקירה – Ripple

במסגרת הפרויקט בחנו את האפשרות להיעזר במאמר " Ripple: Communicating through Physical Vibration" [5] ולממש חלקים מההצעות שמופיעות בו. יש לציין שבמאמר נעשה, בין היתר, שימוש ברכיב מד-תאוצה זהה לזה שקיים בחיישן שרכשנו (ADXL345). רכיב זה נפוץ מאוד בטלפונים חכמים.

המאמר מציע פרוטוקול העברת מידע מעל רטט – "Ripple" – תחילה בתוך מערכת ייעודית ולאחר מכן מעל טלפונים חכמים שמריצים אנדרואיד. המערכת הייעודית, אשר מממשת את הפרוטוקול באופן מלא, מורכבת ממנוע רטט וממד-תאוצה אשר נשלטים באמצעות לוח ארדואינו. באמצעות הלוח ניתן לשלוט ברמות המתח שמזין את המנוע, ובכך לשלוט בתדר ובאמפליטודה של הרטט שנוצר. האות עובר באמצעות אפנון אמפליטודה (ASK) על-גבי 10 גלים נושאים במקביל.

לעומת המערכת הייעודית, השליטה במאפייני הרטט מעל אנדרואיד מוגבלת – ניתן לשלוט באמפליטודה באמצעות שימוש ב-API קרנל-י, אך לא ניתן לשלוט בתדר ולכן לא ניתן להשתמש ביותר מגל נושא אחד בתדר קבוע מראש. על-מנת לשפר את ביצועי מערכת זו נעשה שימוש במתקן עשוי עץ עליו מונח המכשיר המשדר.

בנוסף לפרטי מימוש השיטה, המאמר מציג שיטות לביטול ולמיסוך הרעש שיוצר מנוע הרטט בזמן הפעולה שלו מטעמי אבטחה, לשם הגנה מפני פענוח המידע על-ידי מאזין קרוב. אין לנו כוונה במסגרת הפרויקט להתייחס לחלק זה במאמר.

4. אתגרים ואילוצים

א. <u>בעיות תזמון מעל ProjectZero</u>

על-מנת לעבד את הרטט ולפענח אותו לכדי אות בעל משמעות הדגימה של הנתונים הגולמיים צריכה להתבצע בקצב גבוה מספיק. בנוסף, עיבוד האות מצריך עיבוד נתונים מסוג float וביצוע פעולות כפל וחילוק ביניהם (בכדי לחשב את השונות עבור חלון הדגימה, 8 או 16 דגימות גולמיות). ניסיונות לשילוב קטע הקוד שמבצע את הדגימה בתוך ProjectZero (פירוט בהמשך) לא צלחו:

- **שילוב כ- Task נפרד**: ב-ProjectZero קיימים מספר Task-ים בעלי עדיפויות שונות (Task-ים בשימוש ICall, Task עבור GAP, Task עבור ייצוא שירותי GATT). הוספה של קוד הדגימה כ- Task נפרד פגעה בפונקציונליות של הפרויקט כולו – במקרים מסוימים אף מנעה עלייה תקינה של הפרויקט.
- **שילוב כ-Idle Task**: הביא לתדר דגימה נמוך מאוד, כלומר רק מספר בודד של דגימות בכל כמה שניות. דגימה כזו לא אפשרה חילוץ נתונים.
- **קריאה לקטע הקוד מתוך Task קיים**: הוספת קריאה לקוד שלנו מתוך ה-Task הראשי של הפרויקט, אשר מייצא את שירותי ה-GATT (ProjectZero_taskFxn). גם כאן נתקלנו בקצב דגימה לא מספק.

בכדי לוודא שהחישובים שאנחנו מבצעים הם לא הגורם שמאט אותנו, ניסינו לשלב את קטע הקוד ללא כל עיבוד, חישוב או מניפולציה על הנתונים עצמם. גם בצורה זו לא הצלחנו להגיע לקצב דגימה מספק.

בגלל חוסר היכולת לשלב בין שני הפרויקטים החלטנו להשאיר אותם בנפרד – פרויקט אחד לתקשורת עם החיישן ופרויקט נוסף לממשק BLE.

ב. <u>אי-יכולת לשלוט באמפליטודת התדר</u>

הצד המשדר במערכת שנבנתה עבור הפרויקט הוא סמארטפון של אנדרואיד, כאשר יצירת תבניות הרטט בוצעה באמצעות אפליקציה חיצונית קיימת (Vibrate Pattern Maker). השימוש באפליקציות קיימות מאלץ שליטה בזמני הפעלה ועצירה בלבד ולא מאפשר שליטה באמפליטודה. השליטה רק במאפיין אחד של התדר, מגבילה אותנו לשימוש בשיטות קידוד בסיסיות אשר מאלצות רוחב פס נמוך ולא מאפשרות את מימוש Ripple.

ג. <u>העברת הרטט מהטלפון אל החיישן ע"י הצמדה</u>

תחילה הנחנו את הטלפון מעל משטח רך, כמו גליל נייר טואלט או כרית, ועליו את החיישן. זה עבד חלקית, אבל לא מספיק טוב.

בשלב השני נעשה שימוש בגומייה בכדי להצמיד את המכשירים – מה שהוכח כטוב יותר אמפירית.

5. תהליך עבודה ופרטי מימוש

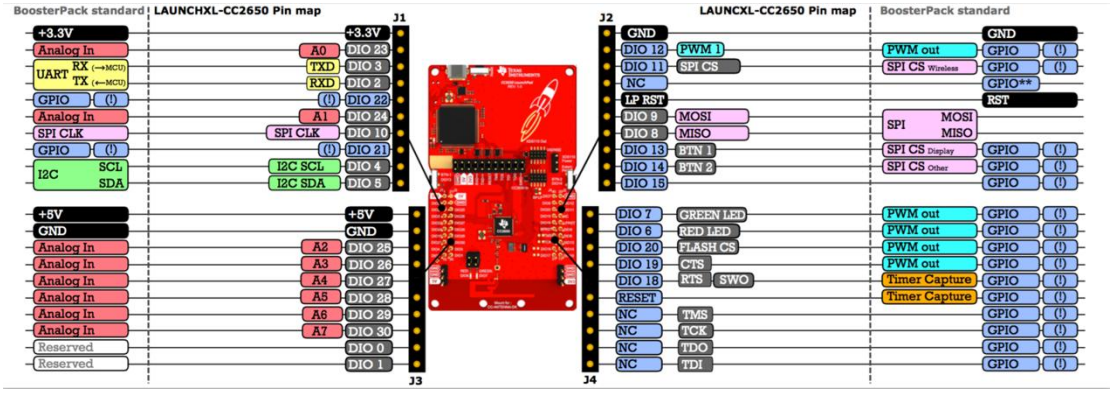
כאמור, המימוש הופרד לשני פרויקטים אשר רצים מעל CC2650:

א. **פרויקט לתקשורת עם החיישן, דגימה ועיבוד אות** – חלק זה כולל את המימוש לתקשורת עם החיישן בפרוטוקול I^2C, ומאפשר קריאה וכתיבה לרגיסטרים של החיישן. בנוסף, ממומש תהליך הדגימה עצמו, בו נקבע גודל חלון הדגימה שעבורו מנותחים הנתונים הגולמיים שנאספים. לבסוף, עיבוד הנתונים הגולמיים והפיכתם לאות בעל משמעות רצויה, כלומר מערך של ערכים לוגיים שמהם ניתן לקודד את הסיסמה. זהו למעשה רצף של '0' ו-'1' לוגיים.

ב. **פרויקט לממשק BLE** – חלק זה כולל מימוש של ומימוש של שירות BLE (GATT service) לממשק לקריאת הנתונים מהחיישן. השירות מתבסס על ה-BLE stack, כפי שממומשת ב-ProjectZero מתוך SimpLink Academy. תוספת נוספת ל-ProjectZero הייתה Authenticated Pairing, התניית התחברות בהזנת ה-PIN code שהתקבל מעל רטט (6 ספרות), מתוך IdleTask ולפני אתחול ה-stack. תהליך התקשורת ב-BLE התבצע מתוך סמארטפון אנדרואיד באמצעות האפליקציה BLE scanner.

5.1. חיבור חיישן GY-801 ללוח CC2650

החיישן דורש חיבור למתח אספקה של $5/3.3_{[V]}$ ואדמה GND. בנוסף ישנם 2 פינים לחיבור בפרוטוקול תקשורת I^2C, שעון (SCL) ומידע (SDA). את הפינים יש לחבר למקומות המיועדים על גבי לוח הפיתוח כפי שניתן לראות בתרשים הבא:



תרשים 2: CC2650, GPIO השונים. עבור חיבור בפרוטוקול I2C, יש להשתמש בפינים DIO 4,5, מסומנים בירוק בהיר.

בנוסף לחיבור הפיזי, יש לממש את התקשורת עם הרכיבים על גבי הלוח ולעדכן קונפיגורציות בפרויקט בכדי לתמוך בתקשורת I^2C. מימשנו בתוכנה קריאה וכתיבה מרגיסטרים של החיישנים בפרוטוקול תקשורת זה. בשלב זה, היה צורך לקחת את הרכיבים למעבדת אלקטרוניקה, ולחבר את קווי השעון והתקשרות לאוסילוסקופ, בכדי לבצע איתור תקלות, ובכדי לוודא שהסיגנלים על

ה-Bus תואמים את התיאוריה.

היות ומדובר בפרוטוקול I^2C, בו יש לפעמים חוסר עקביות בתיעוד הכתובת של החיישן וביחס לקריאה/כתיבה, היינו צריכים לוודא שהכתובות שמצוינות בדפי המידע (Datasheet), הן בפורמט המתאים לזה שהלוח מדבר בו. שלב זה אכן היה מאתגר מעט, היות ואכן לא הייתה התאמה בשלב הראשון.

לאחר שהשגנו קישוריות מלאה, ניתן היה לבחור פרמטרים מתאימים לכל חיישן על הלוח.

החיישנים מאפשרים פעולה ברזולוציות שונות, תדרי דגימה שונים, ובעלי תכונות (features) מיוחדות שאותן יש לבטל/לאפשר על פי הצורך (למשל – זיהוי נפילה חופשית במד-התאוצה). כמו כן החיישנים תומכים במצבי פעולה/שינה בכדי לאפשר חיסכון בהספק הנצרך.

בשלב הבא, ביצענו מימוש של פענוח הזויות במרחב (Yaw, Pitch, Roll) ע"י תוכנה, בכדי לוודא שאכן החיישנים שלנו מציגים מידע אמין, כלומר שהרכיבים הגיעו אלינו תקינים.

5.2. העברת מפתח הצפנה ברטט ופענוחו

1. חישוב זוויות Pitch, Yaw, Roll כשלב ראשון של אינטגרציה עבור הרכיב לתוכנה שלנו, ובדיקת רגישות הזוויות המתקבלות אל מול רטט.
2. בחינה מחדש של הנתונים הגולמיים שמתקבלים מהחיישנים, האם יש צירים מסוימים במד התאוצה או הג'ירוסקופ שמאפשרים מידול טוב יותר של הרטט.
3. דגימה מהירה של הנתונים, שבסופה הם נכתבו לקובץ נתונים גולמיים. את הקובץ הזה העברנו בתחילה לMS-Excel, ובהמשך ל Matlab, בכדי שנוכל לבצע אנליזה ועיבוד אות.

עיבוד התוצאות הוביל ל-2 מסקנות עיקריות.

1. יש חריגים (outliers) במדידות. התמודדנו עם החריגים ע"י החזקה של קבוצה סופית של המדידות הגולמיות האחרונות ובחירה של הערך המינימאלי מבין 3 המדידות הגולמיות האחרונות בכל איטרציה, זה הוריד את החריגים לאפס, משום שהחריגות היו בכל פעם במדידה אחת ואף פעם לא ברצף.
2. שימוש בשונות (Variance) של ערך המדידות יביא להבחנה טובה ומספקת עבור רטט/שקט. למעשה אנו מבצעים חישוב של שונות עבור כל ציר בנפרד, ולאחר מכן מבצעים סכימה של השונות בשלושת הצירים. לאחר שמדדנו מספר רב של דגימות, מצאנו מה האחוזון ה99 של השונות של רעשים מהמדידות, וקבענו אותו כ-"סף" (threshold). כך למעשה, אנו יודעים שכל מה שיחצה את האחוזון הזה, הוא אות (signal) שאותו אנו רוצים לבדוק.

בשלב זה מימשנו את המסקנות שלנו מהניתוח ב-Matlab, על גבי ה- Microcontroller.

ביצענו ניסויים נוספים בכדי לוודא שאכן הערכים שבחרנו מתאימים. בנוסף, בדקנו את השפעת גודל חלון הדגימה על השונות וביצענו התאמה של החלקת רעשים על המערכת בפועל. כמו כן החלטנו על אורך של רטט קצר/ארוך, בכדי להתאים רטט קצר ל '0' לוגי, ורטט ארוך

ל '1' לוגי.

5.3 פיתוח מעל BLE

Project Zero מממש באופן מלא את ה-Stack של פרוטוקול BLE וכולל הצפנת מידע לאחר Pairing (AES-128). הפרויקט מאפשר הוספה של שירותים (Service-ים) חדשים ומכיל דוגמאות מפורטות למימוש. ה-API לפיתוח GATT ומשמשת להרצת קוד וניהול GAP (ופיצ'רים נוספים של BLE) כתוב מעל הספרייה ICall שמספקת TI זיכרון מעל TI-RTOS. על-פי התיעוד, ספריה זו גם כוללת לוגיקה לניהול הספק.

אל Project Zero הוספנו את החלקים הבאים:

1. Authenticated Pairing:

במטרה להעניק משמעות לתהליך החלפת המפתח שהתבצע בחלק 2 שונו חלקים באתחול ה-BLE כך שהמפתח שהתקבל בחלק הקודם ישמש כ-PIN code (מכונה ב- SimpleLink Academy – "Passcode"). השינויים שנעשו היו בפרמטרים לאתחול ה- Gap Bond Manager, מודול שאחראי על מימוש היבטי האבטחה המרכזיים בפרוטוקול:

א. GAPBOND_DEFAULT_PASSCODE – שונה להיות המפתח שהתקבל בחלק 2.

ב. GAPBOND_PAIRING_MODE– שונה להיות GAPBOND_PAIRING_MODE_INITIATE (במקום GAPBOND_PAIRING_MODE_WAIT_FOR_REQ) על מנת לאפשר יזימה של בקשת PIN code מצד האפליקציה שלנו.

אתחול ה-Bond Manager, כמו אתחול כל ה-BLE stack, קורה בתחילת ה-Task הראשי של הפרויקט, (הפונקציה ProjectZero_taskFxn), לכן הוספנו סמפור בתחילתו כך שהאתחול יתבצע רק אחרי סיום חלק 2.

2. GATT Service לקריאת נתוני החיישן:

לאחר תהליך ה-Pairing משתמש באפליקציה יכול לקרוא את נתוני החיישן באמצעות GATT service ייעודי (acc_service.c). כמו במקרה של שלושה מתוך ששת ה-Service-ים ש-Project Zero מייצא היום (LedService, ButtonService, DataService), אין לנו אפשרות להשתלב באחד מה-Service-ים של התקן, ולכן ה-Service נכתב כ-Custom Service. (כלל ה-Service-ים הנתמכים מוגדרים בקובץ gatt_profile_uuid.h).

Attributes ב-Service (מאותה סיבה, גם מוגדרים כ-Custom):

Service Name **not visible via app**	UUID	Properties
Pitch	F0004441-0451-4000-B000-00000000000000	READ,NOTIFY
Yaw	F0004442-0451-4000-B000-00000000000000	READ,NOTIFY

ה-Service מייצא את הפונקציה ACC_Service_WriteAttr_App התומכת בשינוי ערכי characteristics של ה-service, וכך מאפשרת (תיאורטית) לקטע הקוד שמתקשר עם החיישן לשנות את הנתונים שהמשתמש רואה, מתוך הפרויקט עצמו. עדכון הנתונים נעשה תוך שימוש ב-API של Project Zero – enqueue ל-message באמצעות הפונקציה user_updateCharVal. ה-message נפתחת א-סינכרונית בלולאת הריצה הראשית ב-ProjectZero_taskFxn.

גודל ערכי characteristics שונה להיות 4 בתים על-מנת לתמוך בערכי float אשר מתקבלים מהחיישן.

3. Idle Task לתקשורת עם החיישן:

בהינתן האילוץ שהזכרנו, התקשורת לא באמת מתבצעת מעל Task זה. ה-Task מכיל stubs (get_code(), update_values()) שבעתיד יכולים לשמש כקריאות לפונקציות התקשורת עם החיישן. ה-Task מבצע post לסמפור ש-Project Zero מחכה עליו לפני שמתחיל את אתחול ה-stack. ה-pend אמור להתבצע אחרי קבלת המפתח.

6. הערכת איכות המערכת וקביעת פרמטרים

1. תדר העבודה עבור חיישן ADXL345 צריך להיות גבוה מתדר ברירת המחדל (100Hz) בכדי לעבד את הנתונים לכדי אות משמעותי, בחרנו לעבוד בתדר 800Hz.

2. הערכים הבאים נקבעו ב- Matlab והותאמו אמפירית לאחר מכן עפ"י קצב הדגימה בפועל, הם למעשה קובעים את פענוח האות מהנתונים הגולמיים:

- SEGMENT – גודל החלון עבורו אנו מבצעים חישוב שונות (Variance) כולל 16 דגימות גולמיות בעבור כל ציר.
- SAMPLES – אורך הבאפר המחזורי שמחזיק את התוצאה של חישובי השונות . עבור כל חלון מבוצע חישוב השונות לכל ציר בנפרד, לבסוף מתבצעת סכימה של השונות בשלושת הצירים, כל תוצאת סכימה נשמרת באחד התאים ונמחקת לאחר SAMPLES דגימות נוספות.
- THRESHOLD – הסף שנקבע עבור שונות שאינה מכילה רטט, אלא רעש בלבד, ביחס לגודל החלון ולתדר הדגימה.
- SHORTVIBEMAX – מספר המדידות שקובעות את ההפרדה בין רטט קצר לארוך (מעין מקסימום + דלתה עבור הרטט הקצר).
- BER – כמות הביטים שניתן להחשיב כטעות בעבור זמן ההפרדה בין רטט אחד למשנהו. לדוגמה, עבור קלט של 1101 עם BER=1 נקבל שרשרת רציפה של '1'.
- SEPERATEVIBE – אורך זמן ההפרדה בין רטטים שונים, זהו מינימום של מספר תוצאות השונות.

Measured time per I2C transaction	22[clk ticks]
SEGMENT	16
SAMPLES	200
THRESHOLD	0.15
SHORTVIBEMAX	160
SEPERATEVIBE	10
BER	3

7. קבצים

פרויקט - project_zero_app_cc2650launchxl:

Startup\main.c	קובץ ה-main המקורי של הפרויקט, בתחילת פונקציית main נוסף אתחול הסמפור שעליו מחכה ה- task הראשי של הפרויקט, לפני אתחול ה-BLE.
Application\project_zero.h Application\project_zero.c	קבצים מקוריים של הפרויקט, נוסף אתחול acc_service לתמיכה בייצוא נתוני החיישן כשירות BLE חדש.
PROFILES\acc_service.h PROFILES\acc_service.c	מימוש GATT service לייצוא נתוני החיישן
Application\project_acc.h Application\project_acc.c	מכיל פונקציה הנרשמת ל-Idle Task. מכיל stubs לתקשורת עם החיישן, לקבלת code ולכתיבת ערכי characteristics בזמן שה-BLE למעלה.

פרויקט - Vibe_adxl345:

swi.c	קובץ ה-main המקורי של הפרויקט, לקובץ זה נוספו הקוד שמבצע את עיבוד הנתונים והקריאה של הנתונים הגולמיים של החיישן.
gy801.c gy801.h	קבצים שמכילים את הפונקציות, הקבועים, וההגדרות של ביצוע פעולות על החיישנים שנמצאים על הלוח GY-801.
i2c_interface.c i2c_interface.h	מימוש תקשורת I2C לקריאה וכתיבה של נתונים מהחיישן.

8. סימוכין

[1] L3G4200D datasheet

[2] ADXL345 datasheet

[3] HMC5883L datasheet

[4] BMP180 datasheet

[5] Roy, Nirupam, Mahanth Gowda, and Romit Roy Choudhury. "Ripple: Communicating through Physical Vibration." *NSDI*. 2015.

[6] "SWRU393_CC2640_BLE_Software_Developer's_Guide", SimpleLink Academy

[7] BLE lessons under C:\ti\simplelink_academy_01_11_00_0000\modules