דוח פרוייקט מסכם – מעבדת משובצות

Tug Of War

מרצה: מעוז לאונץ

עוזר מעבדה: אדם סופר

רון אלטר	תמרה בייבצ'וב	רון גרשבורג	מתן כהן
200836393	308208936	313164766	204332910

January 2020

1. דרישות לקוח:

הפרויקט מבוסס על המשחק Tug Of War (משיכה בחבל), באמצעות בקר ה-MSP430.

פעולת ה"משיכה בחבל" תתבצע באמצעות כפתורי ה-GROVE שחיברנו לבקר.

משך המשחק נקבע ע"י המשתמשים:

- הכנסת הערך 1 – אורך המשחק הנבחר הינו 10 שניות.
- הכנסת הערך 2 – אורך המשחק הנבחר הינו 15 שניות.
- הכנסת הערך 3– אורך המשחק הנבחר הינו 20 שניות.

לאחר מכן, תופיע הודעת "prepare" למסך (שמטרתה להכין את השחקנים לתחילת ההתמודדות) המלווה ב-2 "איתותים" של שתי נורות ה-LED המובנות בבקר.

בשלב זה המשחק מתחיל! שני שחקנים מתחרים ביניהם מי לוחץ מספר רב יותר של פעמים על כפתור ה GROVE במהלך המשחק (פעולה זאת משולה לאיזה שחקן מושך "חזק יותר" בחבל).

במהלך ריצת המשחק, לשחקנים מודפס למסך מי מוביל בכל רגע נתון (בעזרת הדפסות על המסך המדמות את "מצב החבל"). כמו כן, הזמן הנותר לסיום המשחק מוצגת לשחקנים בעזרת "איתותים" מנורות ה-LED המובנות בבקר לפי הלוגיקה הבאה: ככל שנותר זמן **קצר** יותר עד לסיום המשחק, תדירות ה"איתותים" הולכת **וגדלה**.

בסיום המשחק (כשפרק הזמן הנבחר תם או לחליפין שאחד השחקנים הגיע לאחד מגבולות "המגרש") מוצג על המסך הודעה המכריזה על המנצח.

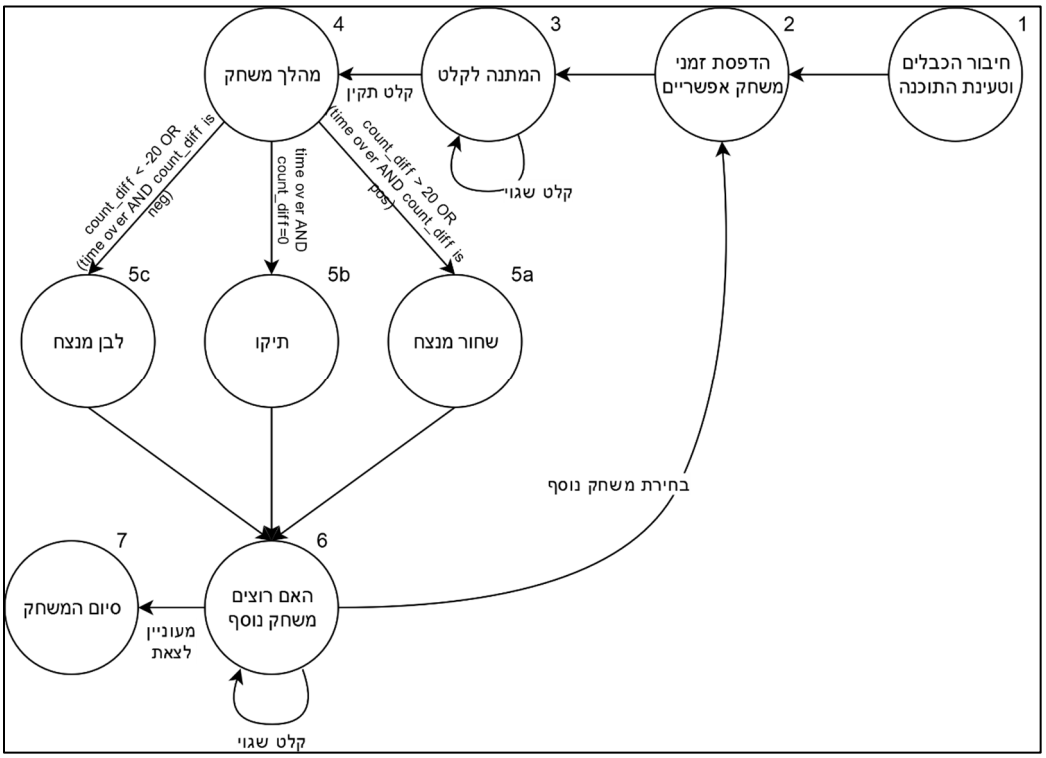
כמו כן, השחקנים יכולים לבחור האם ברצונם להתחיל משחק חדש או לצאת ממנו.

2. מערכת:

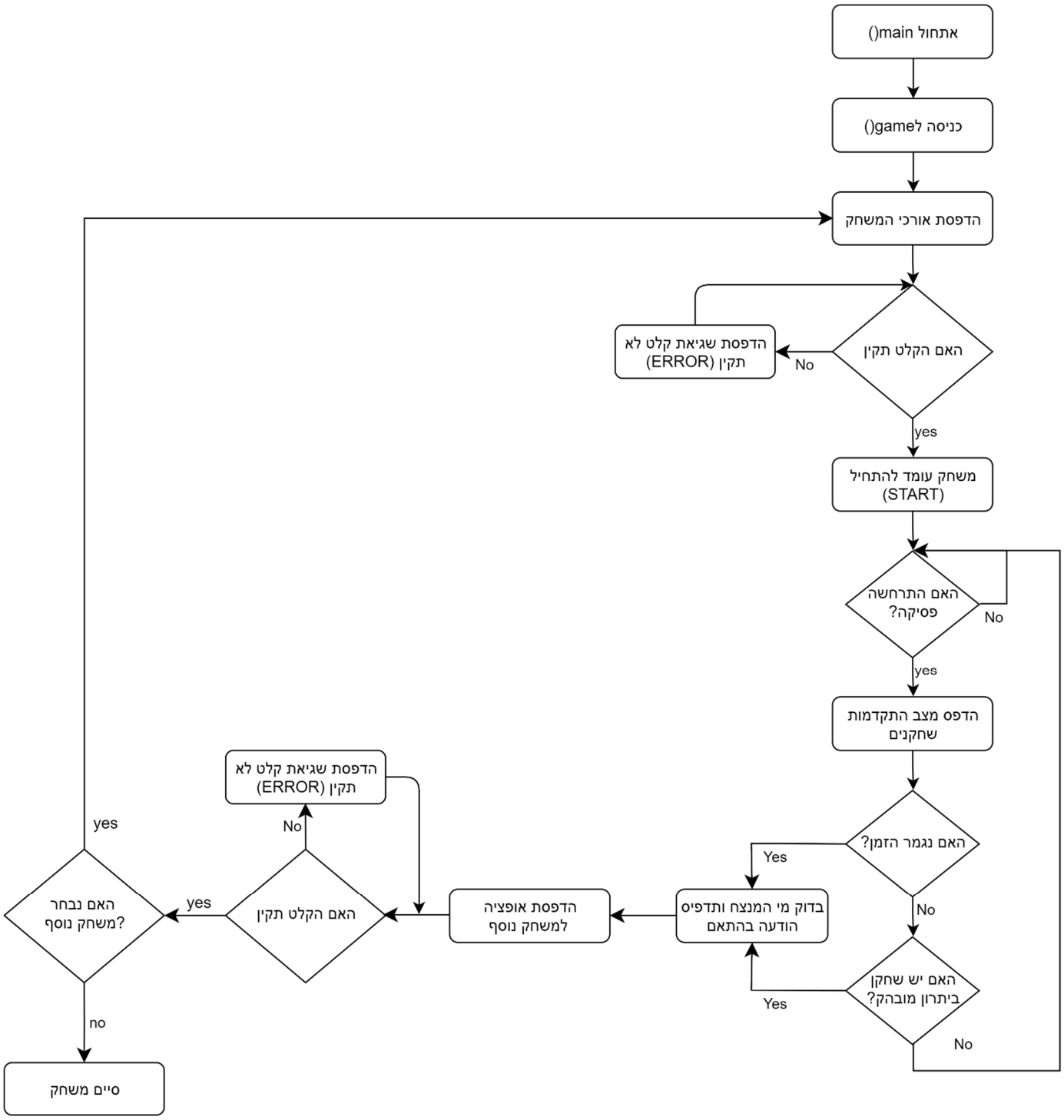
הרכיבים במערכת:

1. בקר MSP430, מחובר למחשב לטובת תקשורת UART עם השחקנים.
2. נורות LED – המערכת כוללת את שתי הנורות המובנות בבקר ה-MSP (ירוק ואדום)
3. כפתורים – המערכת כוללת 2 כפתורי GROVE, אחד עבור כל שחקן.

מערכת מצבים:



תרשים זרימה של המערכת:



מגבלות מערכת:

1. המשחק מוגדר מראש ל3 אורכי משחק שונים.
2. מתח – חיבור $5v$ דרך usb.
3. חיבורים – אורך צמה מוגבלת באורכה, משיכת החיבור יגרום לניתוק מהבקר ונזק לחומרה.
4. טמפרטורה – בקר MSP עובד בטווח $-40°C\ to\ 85°C$.

3. אלגוריתם

לא השתמשנו באלגוריתם מוכן או מוכר למימוש המערכת.

4. הפעלה

סדר פעולות על מנת להפעיל את המערכת:

1. טעינת קובץ הפרויקט לבקר אשר מחובר בחיבור USB למחשב.
2. חיבור חומרה

חיבור כפתור grove1:
GND - חיבור לPin1
VCC - חיבור מקבילי לPin2 (דרך מתאם)
Sig - חיבור לPin3
חיבור כפתור grove2:
GND - חיבור לPin12
VCC - חיבור מקבילי לPin2 (דרך מתאם)
Sig - חיבור לPin4

3. הגדרת חיבור UART בטרמינל לטובת תקשורת עם הבקר.

מה צריך לצפות (כללי המשחק/מעבר משלב לשלב):

כללי המשחק הם פשוטים, לאחר בחירת אורך המשחק על ידי המשתמשים המשחק מתחיל באופן מעשי.

המטרה של כל אחד מהשחקנים היא ללחוץ על כפתור המשחק שלו יותר פעמים מיריבו בפרק הזמן הנבחר.

המשחק יסתיים כאשר אחת משתי האפשרויות הבאות תתממש:

- אחד השחקנים "משך את החבל עד לקצה שלו".
- משך זמן המשחק שהוגדר על ידי המשתמשים הסתיים.

לאחר שאחת משתי האפשרויות מתממשות המשחק מסתיים ומתגלה זהות המנצח.

מה רואים במסך:

בהפעלה ראשונה של התוכנית תופיע על המסך הדפסה (פלט) המבקשת מהשחקנים לבחור את אורכו של המשחק (פירוט על האפשרויות המוצעות - ראו כותר "דרישות לקוח").

לאחר שהמשתמש בוחר באחת מהאפשרויות, מוצגת הודעה נוספת אשר מודיעה על אפשרות המשחק הנבחרת ע"י המשתמשים, וכמו כן, על כך שהמשחק עומד להתחיל.

לאחר מכן תחל ספירה לאחור – אשר מזוהה בעזרת הנורות שעל הבקר – כאשר הן נכבות בפעם האחרונה המשחק מתחיל. (תחילה הן כבויות, נדלקות-> נכבות, נדלקות->נכבות +משחק מתחיל!).

במהלך המשחק יודפס "מצב החבל" למסך (המהווה "אינדיקציה" לתוצאת המשחק באותו רגע נתון) בכל שנייה אחת.

במידה ועברו 5 שניות במהלכן אף אחד מהמשתמשים לא לחץ על הכפתור ("לא שיחק"), תודפס הודעה למסך:

"Are you here?".

בסיום המשחק תוכרז זהות המנצח, ולמשתמשים תעלה האפשרות להתחיל משחק חדש או לסיימו.

5. קוד ופונקציות

1. main()
קלט:אין
פלט:אין
תפקיד: אתחול המערכת (ports,interrupts,UART ,TIMER),קריאה לפונקציה game().
גודל זכרון: $34B$

2. game()
קלט: אין
פלט: אין
תפקיד: מנהלת את המשחק (הדפסת התפריט למסך, קבלת ערכי הקלט מהמשתמשים בנוגע לאורך המשחק, התחלת משחק חדש/סיום המשחק).
גודל זכרון: $274B$

3. init()
קלט: אין
פלט: אין
תפקיד: מבצעת איפוס למשתנים הגלובליים בהם אנו משתמשים במהלך המשחק.
גודל זכרון: $56B$

4. interrupt void Port2(void)
קלט: אין
פלט: אין
תפקיד: מחשבת תוצאת המשחק בכל רגע נתון (כל לחיצה על כפתור על ידי אחד המשתמשים מעדכנת את ערך המשתנה count_diff).
גודל זכרון: $200B$

5. interrupt void Timer_A(void)__
קלט: אין
פלט: אין
תפקיד: מטפלת בפסיקות שמתקבלות מ-timer_A. אחראית על הדלקה וכיבוי הנורות (לסירוגין) טרם תחילת המשחק (תפקיד הנורות בשלב זה הוא להכין את המתמודדים לקראת תחילת המשחק).
במהלך המשחק, בכל פסיקה נבדוק האם אחד המשתמשים "ניצח" על פי חוקי המשחק.
כמו כן, פסיקה זאת אחראית על הדפסת "מצב החבל" במהלך המשחק.
גודל זכרון: $382B$

6. interrupt void USCI0RX_ISR(void)__
קלט: אין
פלט: אין
תפקיד: בדיקת נכונות הקלט של המשתמש, אתחול המשתנים לקראת משחק חדש.
גודל זכרון: $204B$

7. interrupt void USCI0TX_ISR(void)__
קלט :אין
פלט: אין
תפקיד: פסיקה זאת מכילה את ה-states הבאים:
- Error – מדפיס למסך הודעת שגיאה ומאפשר קבלת קלט חדש מהמשתמש.
- Start- מדפיס את האפשרות זמן המשחק שנבחרה על ידי המשתמש והודעה על כך שמהשחק עומד להתחיל.
- Endgame – בודק את זהות המנצח ומדפיס "הודעת ניצחון" בהתאם.

גודל זכרון: $234B$

6. זיכרון

גודל זיכרון ה- Flash בתוכנית שלנו (ללא אופטימיזציה) הוא: 2.4KB ~ 2391B

RAM	326 (31%)	1,024
FLASH	2,391 (7%)	32,734

גודל זיכרון ה- RAM בתוכנית שלנו (ללא אופטימיזציה) הוא: $600B$

Memory Browser | Disassembly

0x200

0x200 <Memory Rendering 5>

16-Bit Hex - TI Style

```
0x0200 2031 7369 7320 6C65 6365 6574 2164 4720 6D61 2065 7369 6120 6F62 7475 7420 206F 6562 6967 2C6E 7020
0x0228 6572 6170 6572 2E2E 0A0D 0000 2D5B 2D2D 2D2D 2D2D 2D2D 2D2D 2D2D 2D2D 2D2D 2D2D 7C2D 2D2D 2D2D 2D2D
0x0250 2D2D 2D2D 2D2D 2D2D 2D2D 2D2D 2D2D 0D5D 0D0A 000A 6C50 6165 6573 6520 746E 7265 6120 7620 6C61 6469
0x0278 6E20 6D75 6562 2172 0D20 000A 6854 2065 6977 6E6E 7265 6920 2073 6C62 6361 216B 2021 0A0D 0000 6854
0x02A0 2065 6977 6E6E 7265 6920 2073 6877 7469 2165 2021 0A0D 0000 7241 2065 6F79 2075 6568 6572 3F3F 0A0D
0x02C8 0000 7449 7327 6120 7420 6569 0A0D 0000 0000 0001 0015 0000 0000 0000 000A 000F 0014 0001 0000 0000
0x02F0 0000 0000 0000 FD4F FFDE 9BBF BDDF EF5F 81A0 EAB5 20A8 AE90 A016 0C00 8000 084A 1000 8081 8DC2 0C1E
0x0318 1024 9100 5A60 E020 10A9 4242 A09B 2008 4835 D524 0B10 8112 8260 1042 2A41 2909 86D7 1210 0004 C901
0x0340 0508 710C 0000 0060 A4B8 80BB 62A4 6108 3121 1204 0C06 0A20 5908 9653 0703 D908 0180 0821 143A 8002
0x0368 A660 B458 4219 1181 C10C 0000 4448 3062 0444 9109 2880 6004 FD7F F9CD F0B7 FFF7 F53F A9FF 9BF5 2BF6
0x0390 EFF3 FFBB F6DD FD77 3BF3 DFCD EBFD E7FB EFFF FECB FFF8 FFCF FFFF DFED 3BF3 CF97 E7FE DFBF B79B F3F1
0x03B8 BDEF 7FFE BFFE DE6B F75F 7F4B E9FE FFFC 9EF3 AFD7 F7F3 E6FE FF3F B7F5 BEBB F53B FFD7 F7DD BFFC DFBF
0x03E0 EFFF B55F 15B6 F3F8 F1ED FE4B FEB7 EB44 EE6F 7A7D F7D3 F5AD D6FF DEFF FE9F DF99 0292 8A59 3408 1201
0x0408 4210 0C04 16B7 14C8 0182 2000 00D4 08A0 1A61 C841 000A 0040 5223 8040 3203 8100 2726 038C 0801 0208
0x0430 6019 2E05 0490 1020 9CA1 1000 0219 C27C 1800 4044 4267 0853 C126 C008 0B6A 0141 51C1 1000 80A5 5046
0x0458 0003 0490 2824 4004 1080 1244 1000 80C1 0845 2410 AF0B 5021 04F4 4A86 0334 2002 0121 8C4A 200B 800D
0x0480 FC7F F7EB EFFE 9EF6 EB74 FFAF 53EF FFD6 BBFD FDB7 F7EC 2FDE 7FD4 7FD7 FDE6 D5BA FD93 BEAE F9FF F3FC
```

גודל זיכרון ה-stack בתוכנית שלנו (ללא אופטימיזציה) הוא: $20B$, באופן כללי מבחינת מקום אפשרי בזכרון קל לראות כי גודל ה-stack המקסימלי בתוכנית שלנו $1024B - 600B = 424B$ אופציונליים לשימוש (כאשר $600B$ הוא גודל הRAM).

Disassembly | Memory Browser

0x0200

0x5b0 - 0x0200(+0x3b0) <Memory Rendering 1>

16-Bit Hex - TI Style

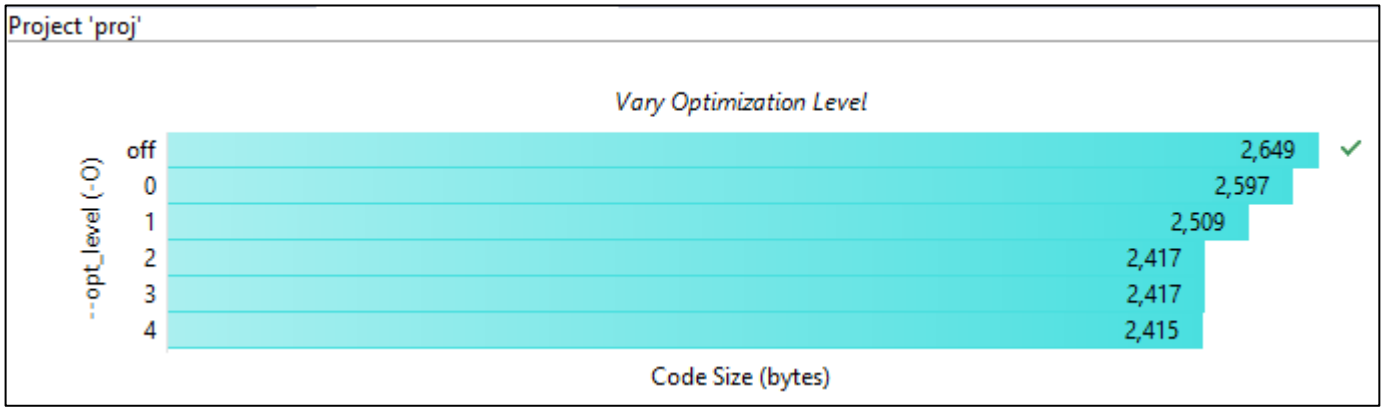
```
0x05B0 _stack
0x05B0 3F6E 0A0D 6F66 2072 2061 656E 2077 6167 656D 7020 6572 7373 3120 0A0D 6F74 6520
0x05D0 6978 2074 7270 7365 2073 0D30 000A 6854 6E61 2[illegible] 2075 202C 6F67 646F 7962
0x05F0 2065 293A 0A0D 0000 0019 0001 85B0 8732 0000 0000 0000 0000 0000 0000 0000 0000
0x0610 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0x0630 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0x0650 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0x0670 0000 0000 0000 0000 0000 0000 0000 894C 0000 0000 0000 0000 0000 0000 0000 0000
0x0690 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0x06B0 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0x06D0 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0x06F0 0000 0000 0000 0000 0000 0000 0000 894C 0000 0000 0000 0000 0000 0000 0000 0000
```

Address 0x5c2

0x5e2

7. אופטימיזציה

בגרף המצורף ניתן לראות כיצד השינויים ברמת האופטימיזציה (opt_level) משפיעים על גודל הקוד (Code Size)
כפי שציפינו, יש יחס הפוך בין רמת האופטימיזציה הנבחרת לבין גודל הקוד:
ככל שאנו **עולים** ברמת האופטימיזציה גודל הקוד הולך **וקטן**.



לא ביצענו שינוי ברמת האופטימיזציה של הקוד, מכיוון שבחישוב מהיר ניתן לראות כי מדובר בשיפור של פחות מ9% וזה אחוז זניח יחסית לעומת האפשרות שתפקוד התוכנה יפגע.
חשוב לציין כי במהלך כתיבת הקוד התייחסנו לייעול הקוד ככל הניתן (למשל לא להשתמש בפונקצית *SizeOf* אלא לחשב בצורה ידנית את גדלי המערכים), בנוסף ביצענו במהלך העבודה על הפרוייקט בדיקות נוספות על מנת לבדוק כיצד ניתן לייעל את הקוד ולצמצם את השימוש בזכרון, תהליכים מסויימים נראו כלא כדאיים בעלות שבין זמן הריצה לעומת גודל הזכרון.

8. צריכת זרם

במהלך התוכנית המערכת עוברת בין power modes שונים.
המצבים בהם נעבוד ב-active mode הם:

- מצב 2 – הדפסת מצבי משחק אפשריים.
- מצב 4 – מהלך המשחק.

המצבים בהם נעבוד ב-LPM3 mode הם:

- המתנה לקלט מהמשתמש (בחירת אורך המשחק הרצוי)
- המתנה לקלט מהמשתמש (האם רוצים משחק נוסף?)

תדר ה-DCO בו בחרנו להשתמש בתוכנית שלנו
במצב active mode הוא: 1MHZ
כפי שניתן לראות מהגרף המצורף (Figure 2), עבור תדר זה –
צריכת הזרם במצב active mode הינה: 0.5mA לערך.

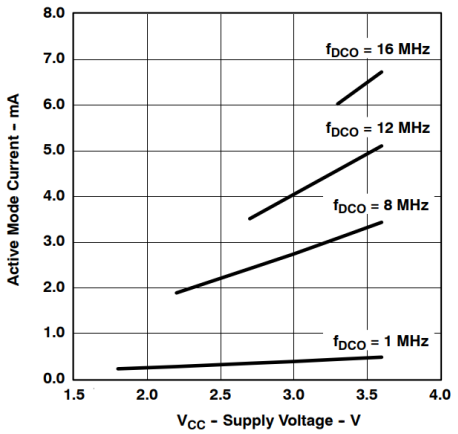


Figure 2. Active Mode Current vs V_{CC}, T_A = 25°C

המעבר מ-active mode ל-LPM3 mode תסייע לנו בחיסכון בצריכת הזרם עבור המשאבים הבאים:

- CPU
- MCLK
- SMCLK
- DCO
- DC generator

Mode	CPU and Clocks Status
Active	CPU is active, all enabled clocks are active
LPM0	CPU, MCLK are disabled SMCLK , ACLK are active
LPM1	CPU, MCLK are disabled, DCO and DC generator are disabled if the DCO is not used for SMCLK. ACLK is active
LPM2	CPU, MCLK, SMCLK, DCO are disabled DC generator remains enabled ACLK is active
LPM3	CPU, MCLK, SMCLK, DCO are disabled DC generator disabled ACLK is active
LPM4	CPU and all clocks disabled

כפי שניתן לראות מהטבלה המצורפת, צריכת הזרם במצב LPM3 mode הינה: 0.9uA (כאשר Vcc=3V,
אנו נמצאים בסביבת טמפרטורת החדר ($25°C$) ואנו משתמשים בבחירת במצב אופייני (TYP)).

$I_{LPM3,LFXT1}$	Low-power mode 3 (LPM3) current, see Note 4	$f_{DCO} = f_{MCLK} = f_{SMCLK}$ = 0 MHz, f_{ACLK} = 32,768 Hz, CPUOFF = 1, SCG0 = 1, SCG1 = 1, OSCOFF = 0	-40°C	2.2 V	0.7	1.4	µA
			25°C		0.7	1.4	
			85°C		2.4	3.3	
			105°C		5	10	
			-40°C	3 V	0.9	1.5	
			25°C		0.9	1.5	
			85°C		2.6	3.8	
			105°C		6	12	

מצב	תיאור המצב	Power mode	משך הזמן	צריכת הזרם
2	הדפסת זמני משחק אפשריים	Active mode	הזמן שלוקח לבצע הדפסה למסך – T2	0.5mA * T2
3	המתנה לקלט מהמשתמש (בחירת אורך המשחק הרצוי)	LPM3	ללא הגבלה (עד לקבלת קלט) – T3	0.9uA * T3
4	מהלך המשחק	Active mode	כאורך המשחק הנבחר על ידי המשתמש – T4	0.5mA * T4
6	המתנה לקלט מהמשתמש (האם רוצים משחק נוסף?)	LPM3	ללא הגבלה (עד לקבלת קלט) – T6	0.9uA * T6

9. בעיות בהן נתקלנו:

- **בעיה:** ריטוטים של מפסק מכני - תקלה ידועה, דיברנו עליה רבות במהלך הקורס (לחיצה בודדת על הכפתור מתפרשת כמספר רב של לחיצות).
  פתרון: על מנת לפתור את בעיית הריטוטים של הכפתור המובנה של הבקר, השתמשנו בכפתור חיצוני (מערכת grove) כאשר כל לחיצה בודדת מעבירה מתח לפורט המתאים בבקר.
- **בעיה:** ניקיון מסך - רצינו שלאחר כל הדפסה על המסך, המסך יתנקה מההדפסות הקודמות שלו.
  פתרון: לא הצלחנו לפתור את בעיה זו בדרך אלגנטית, מכיוון שאנו מעבירים ערכים ASCII-ים למסך, אנו לא יודעים כיצד ניתן להעביר פקודת "clean screen" דרך פרוטוקול סיריאלי, חיפוש בגוגל לא עזר במקרה זה, חשבנו "לדחוף" המון "אנטרים" למסך, הבנו שזה לא יעיל, לאחר מספר הרצות של התוכנית הבנו שבעיית הניקיון מסך זניחה, והנראות של התוכנית לא נפגמת מכך שהמסך לא מתנקה.
- **בעיה:** מגבלת מרחק מהבקר - ג'אמפרים קצרים (כ-15 ס"מ) גורם לחוסך נוחות במהלך התפעול.
  פתרון: שרשור חיבורים "ארוכים" יחסית על מנת לאפשר מרחק סביר בין הכפתורים ונוחות מרבית לשחקנים.
- **בעיה:** קיים רק חיבור מתח אחד (קיים רק פורט Vcc אחד ביציאה מהבקר).
  פתרון: חיבור מפצל.

10. בדיקת תקינות למערכת:

1. קלטים תקינים: המערכת עלולה להגיב בצורה לא רציונלית, ולשנות את מצבה עקב קלט לא תקין או "להיתקע" במצב בו היא נמצאת (כלומר לא להמשיך בריצתה התקינה של התוכנית).
   פתרון: ביצוע בדיקה שהקלט אכן תקין. במידה והתקבל קלט לא תקין מוצגת הודעת שגיאה ומערכת מחכה לקלט חדש (המצב ההתחלתי של אותו state).
2. קבלת קלט בזמן לא צפוי: המערכת עלולה להגיב לקלט לא צפוי, לשנות את מצבה ואף להרוס מהלך תקין של משחק.
   פתרון: הפסקת קבלת פסיקות RX על מנת שהמערכת לא תגיב לקלט מהמשתמש בזמן משחק, ובזמנים שהמערכת לא מצפה לקבל קלט מהמשתמש.
3. וידוא סיום משחק ועצירת תוכנית: לאחר פקודת סיום (הקשה על הערך 0 בתפריט סיום המשחק), המערכת צריכה לצאת מהמשחק ולעצור אותו.
   פתרון: לאחר קבלת פקודה לסיום המערכת נכנסת למצב "שינה" – LMP4 (פירוט על האופציונליות של מצב זה – ראו כותר "צריכת זרם".

11. בדיקות וQA:

1. חיבור בסיסי של כפתור אחד למערכת, וידוא תפקוד עם פסיקה בצורה תקינה.
2. נעשתה בדיקת קלט בנפרד עבור כל אחד מן הכפתורים. וידוא שהם מגיבים בנפרד ומבצעים את הנדרש (מעביר מתח לכניסה הרצויה בבקר).
3. בדיקות תקינות בסיסיות:
   1. הרצת התוכנה עבור זמן לא מוגבל, ווידוא שכל לחיצה על אחד הכפתורים מחשבת את תוצאת המשחק כנדרש.
   2. בדיקות וידוא קלט ומעבר נכון בין מצבי המערכת שונים.
   3. וידוא שאין אפשרות להזין קלט כלשהו במהלך ריצת המשחק.
   4. וידוא שהמשחק עוצר כשהזמן נגמר כפי שהוגדר על ידי המשתמשים בתחילת המשחק.
   5. וידוא שהמשחק מסתיים לפני תום הזמן המבוקש במידה ויש משתמש שהגיע לגבול המוגדר של המשחק.
   6. וידוא שקבלת הקלט מהלחצן נעשית בזמן שהוגדר בתוכנית (רק לאחר סיום ההבהוב הראשוני וכל לחיצה שנעשית לפני לא משפיעה על תוצאות המשחק).
   7. וידוא שלחיצה אחת לא גורמת למערכת לחשוב שיש ריבוי לחיצות (בעיה נפוצה בכפתור המובנה של ה-MSP).
4. בדיקת ניצול זיכרון בצורה מיטבית (במקום להשתמש בפונקציה המובנית sizeof() אשר נכחנו לגלות כי הינה בזבזנית מבחינת מקום, בחרנו ליצור מערך בגודל קבוע (שנקבע כגודל המערך הנקבע מראש)).
5. בדיקה אופן כתיבת זהות המנצח למסך ("The winner is black/white" או לחלופין הגדרת משתנה בשם "white" ומשתנה נוסף בשם "black").

לאחר שביצענו את השוואה גילינו כי כאשר השתמשנו במערך יחיד הכולל את זהות המנצח, ניצול ה-RAM גבוה יותר אך ניצול ה-FLASH נמוך יותר לעומת השיטה השנייה.

12.הקוד של התוכנית

לצורך הפעלת המערכת, חשוב מאוד להצטייד בציוד המתואר בסעיפים הקודמים ולפעול על פי ההנחיות שנכתבו בסעיף 4. מלבד זאת, מומלץ מאוד מאוד להפעיל את תוכנת הטרמינל רק לאחר כניסה למצב debug בתוכנת הCCS.

אנחנו השתמשנו בteraterm לאורך הסמסטר ולא בטרימנל המובנה של CCS מטעמי נוחות.

```
#include <msp430.h>
/*
Tug Of War
Ron Alter - 200836393
Tamara Baybachov - 308208936
Matan Cohen - 204332910
Ron Gershburg - 313164766
*/

//States:

#define INIT 0
#define ERROR 1
#define START 2
#define END_GAME 3

#define DIFF 20
char error_text[] = "Please enter a valid number! \r\n";
char startingtext[] = "  is selected! Game is about to begin, prepare..\r\n";
char here_text[] = "Are you here??\r\n";
char indicator[] = "[--------------------|--------------------]\r\n\r\n";
char no_one[] = "It's a tie\r\n";
char black[] = "The winner is black!! \r\n";
char white[] = "The winner is white!! \r\n";
volatile int timeCount, blinktimer;
volatile int count_diff, playeralive;
volatile int userInput, endgame = 0, stopgame = 0;
volatile int time[3] = { 10, 15, 20 };
volatile int winnerindex = 21, lastcount_diff;

int j = 0;
int flagStart = 0;
int state = INIT;

// black is for plus - P3 for signal and P1 for GND
// white is for minus - P4 for signal and P12 for GND

void game();
void init();
int main(void){
    WDTCTL = WDTPW + WDTHOLD; // stop watchdog timer

    BCSCTL1 = CALBC1_1MHZ; // make the SMCLK to work exactly in 1MHZ
    DCOCTL = CALDCO_1MHZ; // make the SMCLK to work exactly in 1MHZ
    //Clock Settings
    BCSCTL3 |= LFXT1S_2;     //set clock to work in very-low-power
    TACCTL0 = CCIE;  // TACCR0 interrupt enabled
    TACCR0 = 12000;   // ~ 1 sec
    //UART Settings
    P3SEL |= 0x30; // Pin P3.5 and P3.4 and P3.3 used by USART module
    //UCA0BR1 = 0x00;
    UCA0CTL1 |= UCSSEL_2 + UCSWRST; // use SMCLK, USCI software reset
    UCA0BR0 = 104; // Divider for getting 9600 baud rate
    UCA0MCTL = 0x02; // UCBRSx=1
```

```
    UCA0CTL1 &= ~UCSWRST; //initialize the USCI
    //Led Settings
    P1DIR |= 0x03;  // configure P1.0 and P1.1 as output
    P1OUT = 0;
    //Button Settings
    P2SEL &= ~(BIT0 | BIT1);          // configure ports p2.0/1 as IO
    P2DIR &= ~(BIT0 | BIT1); // configure input ports: p2.0=3pin p2.1=4pin on the board
    __bis_SR_register(GIE); //  enable GI
    game();
    UC0IE = 0;
    __bis_SR_register(GIE+LPM4_bits); // enable General interrupt
}

void game(void){
    char text[] = "Lets Play!\r\nplease enter a number:\r\nfor 10 sec game press
1\r\nfor 15 sec game press 2\r\nfor 20 sec game press 3\r\n";
    char play[] = "Do you want to play again?\r\nfor a new game press 1\r\nto exit
press 0\r\n";
    char goodbye[] = "Thank you , goodbye :)\r\n";
    int i, dump;
    init();
    while (stopgame == 0){
        for (i = 0; i < 112; i++){
            while (!(IFG2 & UCA0TXIFG));//wait if the buffer isnt clean
            UCA0TXBUF = text[i];
        }
        UC0IE |= UCA0RXIE;
        __bis_SR_register(GIE + LPM3_bits); //  enable General interrupt + CPU
        while (endgame == 0);
        while((UCA0STAT & UCOE) == UCOE){ //make sure user is not using the keyboard
during the game
            dump = UCA0RXBUF - '0';
        }
        for (i = 0; i < 70; i++){ //play
            while (!(IFG2 & UCA0TXIFG)); //wait if the buffer isnt clean
            UCA0TXBUF = play[i];
        }
        UC0IE |= UCA0RXIE;
        __bis_SR_register(GIE + LPM3_bits);
        while (endgame == 1);
    }
    for (i = 0; i < 25; i++){//goodbye
        while (!(IFG2 & UCA0TXIFG)); //wait if the buffer isnt clean
        UCA0TXBUF = goodbye[i];
    }
    init();
    return (0);
}

void init(void){
    indicator[winnerindex + lastcount_diff] = 45; //make the array ready for a new game
    indicator[winnerindex] = 124; //make the array ready for a new game
    P1OUT = 0;
    flagStart = 0;
    timeCount = 0;
    count_diff = 0;
    lastcount_diff = 0;
    playeralive = 0;
    TACTL &= !MC_1; // shutdown the clock
    P2IE &= ~(BIT0 | BIT1); //make sure P2 will not get interupts
}
```

```
//Buttons
#pragma vector=PORT2_VECTOR
__interrupt void Port2(void){
    if (P2IFG & 0x01)
        count_diff++;
    if (P2IFG & 0x02)
        count_diff--;
    playeralive = 1;
    P2IFG &= 0xFC;
}

//Timer interrupt
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A(void){
    //Code before game starts: makes leds to blink 2 times.(total of 4 sec)
    if (!flagStart){
        P1OUT ^= 0x03;   // turn red and green leds on and off (P1.1+P1.0)
        timeCount++;
        if (timeCount == 4){
            flagStart = 1;
            timeCount = 0;
            P2IFG = 0x00;
            P2IE |= (BIT0 | BIT1);       // configure p2.0/1 enable interrupt
        }
    }
    else{  //Enters while game is on.
        timeCount++;
        if (timeCount == userInput){
            TACTL &= !MC_1;
            state = END_GAME;
            UC0IE = UCA0TXIE; //enable TX interrupt
        }
        if (timeCount % blinktimer == 0){
            P1OUT ^= 0x03;
        }
        if (timeCount % 5 == 0 && blinktimer > 1){
            blinktimer--;
        }
        if (count_diff < -DIFF){ //left side.. white is winning
            indicator[winnerindex + lastcount_diff] = 45;
            indicator[1] = 124;
            lastcount_diff = -DIFF;
        }
        else if (count_diff > DIFF){ //right side.. black is winning
            indicator[winnerindex + lastcount_diff] = 45;
            indicator[41] = 124;
            lastcount_diff = DIFF;
        }
        else{
            indicator[winnerindex + lastcount_diff] = 45;
            indicator[winnerindex + count_diff] = 124;
            lastcount_diff = count_diff;
        }
        for (j = 0; j < 48; j++){
            while (!(IFG2 & UCA0TXIFG)); //wait if the buffer isnt clean
            UCA0TXBUF = indicator[j];
        }
        j = 0;
        if (winnerindex + count_diff <= 1 || winnerindex + count_diff >= 41){
            TACTL &= !MC_1;
            state = END_GAME;
```

```
            UC0IE = UCA0TXIE; //enable TX interrupt.
        }
        if (timeCount == 5 && playeralive == 0){
            for (j = 0; j < 17; j++){
                while (!(IFG2 & UCA0TXIFG))
                    ; //wait if the buffer isnt clean
                UCA0TXBUF = here_text[j];
            }
            j = 0;
        }
    }
}

//RX Uart Interrupt
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void){
    userInput = UCA0RXBUF - '0'; //get input and cast from char to int
    if (endgame == 0){
        if (userInput > 3 || userInput < 1){ //check if input is right.
            state = ERROR;
        }
        else{
            state = START; //print that game starts.
            startingtext[0] = userInput + 48;
            userInput = time[userInput - 1];
            blinktimer = (userInput / 5) - 1;
        }
        UC0IE = UCA0TXIE; //enable TX interrupt
    }
    else{
        if (userInput > 1 || userInput < 0){ //check if input is right.
            state = ERROR;
            UC0IE = UCA0TXIE;
        }
        else if (userInput == 1){ //start over for a new game
            startingtext[0] = userInput + 48;
            userInput = time[userInput - 1];
            blinktimer = (userInput / 5) - 1;
            endgame = 0;
            init();
            UC0IE &= ~UCA0TXIE;
        }
        else{
            endgame=0;
            stopgame = 1;
            UC0IE &= ~UCA0TXIE;
        }
    }
    LPM3_EXIT;
}

//TX Uart Interrupt
#pragma vector=USCIAB0TX_VECTOR
__interrupt void USCI0TX_ISR(void){
    switch (state) {
    case ERROR: //error input state.
        UCA0TXBUF = error_text[j];
        j++;
        if (j == 32){ //108 is number of chars in the message
            j = 0;
            state = INIT; //get back to first stage in order to print new message
```

```
            UC0IE = UCA0RXIE; //enable RX
        }
        break;
    case START: //print users input.
        UCA0TXBUF = startingtext[j];
        j++;
        if (j == 51){ //51 is number of chars in the message
            j = 0;
            state = INIT; //get back to first stage in order to print new message
            UC0IE = 0; // make sure we will not get any interupts
            TACTL = TASSEL_1 + MC_1; // ACLK, upmode
            P1OUT ^= 0x03;
        }
        break;
    case END_GAME:
        if (count_diff > 0){
            UCA0TXBUF = black[j];
            j++;
            if (j == 25){
                j = 0;
                state = INIT; //get back to first stage in order to print new message
                UC0IE = 0; // make sure we will not get any interupts
                endgame = 1;
            }
            break;
        }
        else if (count_diff < 0){
            UCA0TXBUF = white[j];
            j++;
            if (j == 25){
                j = 0;
                state = INIT; //get back to first stage in order to print new message
                UC0IE = 0; // make sure we will not get any interupts
                endgame = 1;
            }
            break;
        }
        else {
            UCA0TXBUF = no_one[j];
            j++;
            if (j == 13){
                j = 0;
                state = INIT; //get back to first stage in order to print new message
                UC0IE = 0; // make sure we will not get any interupts
                endgame = 1;
            }
            break;
        }
    }
}
```

```
 #include <msp430.h>
/*
Tug Of War
Ron Alter - 200836393
Tamara Baybachov - 308208936
Matan Cohen - 204332910
Ron Gershburg - 313164766
*/

//States:
```

```
#define INIT 0
#define ERROR 1
#define START 2
#define END_GAME 3

#define DIFF 20
char error_text[] = "Please enter a valid number! \r\n";
char startingtext[] = "  is selected! Game is about to begin, prepare..\r\n";
char here_text[] = "Are you here??\r\n";
char indicator[] = "[--------------------|--------------------]\r\n\r\n";
char no_one[] = "It's a tie\r\n";
char black[] = "The winner is black!! \r\n";
char white[] = "The winner is white!! \r\n";
volatile int timeCount, blinktimer;
volatile int count_diff, playeralive;
volatile int userInput, endgame = 0, stopgame = 0;
volatile int time[3] = { 10, 15, 20 };
volatile int winnerindex = 21, lastcount_diff;

int j = 0;
int flagStart = 0;
int state = INIT;

// black is for plus - P3 for signal and P1 for GND
// white is for minus - P4 for signal and P12 for GND

void game();
void init();
int main(void){
    WDTCTL = WDTPW + WDTHOLD; // stop watchdog timer

    BCSCTL1 = CALBC1_1MHZ; // make the SMCLK to work exactly in 1MHZ
    DCOCTL = CALDCO_1MHZ; // make the SMCLK to work exactly in 1MHZ
    //Clock Settings
    BCSCTL3 |= LFXT1S_2;     //set clock to work in very-low-power
    TACCTL0 = CCIE;  // TACCR0 interrupt enabled
    TACCR0 = 12000;   // ~ 1 sec
    //UART Settings
    P3SEL |= 0x30; // Pin P3.5 and P3.4 and P3.3 used by USART module
    //UCA0BR1 = 0x00;
    UCA0CTL1 |= UCSSEL_2 + UCSWRST; // use SMCLK, USCI software reset
    UCA0BR0 = 104; // Divider for getting 9600 baud rate
    UCA0MCTL = 0x02; // UCBRSx=1
    UCA0CTL1 &= ~UCSWRST; //initialize the USCI
    //Led Settings
    P1DIR |= 0x03;  // configure P1.0 and P1.1 as output
    P1OUT = 0;
    //Button Settings
    P2SEL &= ~(BIT0 | BIT1);         // configure ports p2.0/1 as IO
    P2DIR &= ~(BIT0 | BIT1); // configure input ports: p2.0=3pin p2.1=4pin on the board
    __bis_SR_register(GIE); //  enable GI
    game();
    UC0IE = 0;
    __bis_SR_register(GIE+LPM4_bits); // enable General interrupt
}

void game(void){
    char text[] = "Lets Play!\r\nplease enter a number:\r\nfor 10 sec game press
1\r\nfor 15 sec game press 2\r\nfor 20 sec game press 3\r\n";
    char play[] = "Do you want to play again?\r\nfor a new game press 1\r\nto exit
press 0\r\n";
```

```
    char goodbye[] = "Thank you , goodbye :)\r\n";
    int i, dump;
    init();
    while (stopgame == 0){
        for (i = 0; i < 112; i++){
            while (!(IFG2 & UCA0TXIFG));//wait if the buffer isnt clean
            UCA0TXBUF = text[i];
        }
        UC0IE |= UCA0RXIE;
        __bis_SR_register(GIE + LPM3_bits); //  enable General interrupt + CPU
        while (endgame == 0);
        while((UCA0STAT & UCOE) == UCOE){ //make sure user is not using the keyboard
during the game
            dump = UCA0RXBUF - '0';
        }
        for (i = 0; i < 70; i++){ //play
            while (!(IFG2 & UCA0TXIFG)); //wait if the buffer isnt clean
            UCA0TXBUF = play[i];
        }
        UC0IE |= UCA0RXIE;
        __bis_SR_register(GIE + LPM3_bits);
        while (endgame == 1);
    }
    for (i = 0; i < 25; i++){//goodbye
        while (!(IFG2 & UCA0TXIFG)); //wait if the buffer isnt clean
        UCA0TXBUF = goodbye[i];
    }
    init();
    return (0);
}

void init(void){
    indicator[winnerindex + lastcount_diff] = 45; //make the array ready for a new game
    indicator[winnerindex] = 124; //make the array ready for a new game
    P1OUT = 0;
    flagStart = 0;
    timeCount = 0;
    count_diff = 0;
    lastcount_diff = 0;
    playeralive = 0;
    TACTL &= !MC_1; // shutdown the clock
    P2IE &= ~(BIT0 | BIT1); //make sure P2 will not get interupts
}
//Buttons
#pragma vector=PORT2_VECTOR
__interrupt void Port2(void){
    if (P2IFG & 0x01)
        count_diff++;
    if (P2IFG & 0x02)
        count_diff--;
    playeralive = 1;
    P2IFG &= 0xFC;
}

//Timer interrupt
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A(void){
    //Code before game starts: makes leds to blink 2 times.(total of 4 sec)
    if (!flagStart){
        P1OUT ^= 0x03;    // turn red and green leds on and off (P1.1+P1.0)
        timeCount++;
```

```
        if (timeCount == 4){
            flagStart = 1;
            timeCount = 0;
            P2IFG = 0x00;
            P2IE |= (BIT0 | BIT1);        // configure p2.0/1 enable interrupt
        }
    }
    else{  //Enters while game is on.
        timeCount++;
        if (timeCount == userInput){
            TACTL &= !MC_1;
            state = END_GAME;
            UC0IE = UCA0TXIE; //enable TX interrupt
        }
        if (timeCount % blinktimer == 0){
            P1OUT ^= 0x03;
        }
        if (timeCount % 5 == 0 && blinktimer > 1){
            blinktimer--;
        }
        if (count_diff < -DIFF){ //left side.. white is winning
            indicator[winnerindex + lastcount_diff] = 45;
            indicator[1] = 124;
            lastcount_diff = -DIFF;
        }
        else if (count_diff > DIFF){ //right side.. black is winning
            indicator[winnerindex + lastcount_diff] = 45;
            indicator[41] = 124;
            lastcount_diff = DIFF;
        }
        else{
            indicator[winnerindex + lastcount_diff] = 45;
            indicator[winnerindex + count_diff] = 124;
            lastcount_diff = count_diff;
        }
        for (j = 0; j < 48; j++){
            while (!(IFG2 & UCA0TXIFG)); //wait if the buffer isnt clean
            UCA0TXBUF = indicator[j];
        }
        j = 0;
        if (winnerindex + count_diff <= 1 || winnerindex + count_diff >= 41){
            TACTL &= !MC_1;
            state = END_GAME;
            UC0IE = UCA0TXIE; //enable TX interrupt.
        }
        if (timeCount == 5 && playeralive == 0){
            for (j = 0; j < 17; j++){
                while (!(IFG2 & UCA0TXIFG))
                    ; //wait if the buffer isnt clean
                UCA0TXBUF = here_text[j];
            }
            j = 0;
        }
    }
}

//RX Uart Interrupt
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void){
    userInput = UCA0RXBUF - '0'; //get input and cast from char to int
    if (endgame == 0){
```

```
        if (userInput > 3 || userInput < 1){ //check if input is right.
            state = ERROR;
        }
        else{
            state = START; //print that game starts.
            startingtext[0] = userInput + 48;
            userInput = time[userInput - 1];
            blinktimer = (userInput / 5) - 1;
        }
        UC0IE = UCA0TXIE; //enable TX interrupt
    }
    else{
        if (userInput > 1 || userInput < 0){ //check if input is right.
            state = ERROR;
            UC0IE = UCA0TXIE;
        }
        else if (userInput == 1){ //start over for a new game
            startingtext[0] = userInput + 48;
            userInput = time[userInput - 1];
            blinktimer = (userInput / 5) - 1;
            endgame = 0;
            init();
            UC0IE &= ~UCA0TXIE;
        }
        else{
            endgame=0;
            stopgame = 1;
            UC0IE &= ~UCA0TXIE;
        }
    }
    LPM3_EXIT;
}

//TX Uart Interrupt
#pragma vector=USCIAB0TX_VECTOR
__interrupt void USCI0TX_ISR(void){
    switch (state) {
    case ERROR: //error input state.
        UCA0TXBUF = error_text[j];
        j++;
        if (j == 32){ //108 is number of chars in the message
            j = 0;
            state = INIT; //get back to first stage in order to print new message
            UC0IE = UCA0RXIE; //enable RX
        }
        break;
    case START: //print users input.
        UCA0TXBUF = startingtext[j];
        j++;
        if (j == 51){ //51 is number of chars in the message
            j = 0;
            state = INIT; //get back to first stage in order to print new message
            UC0IE = 0; // make sure we will not get any interupts
            TACTL = TASSEL_1 + MC_1; // ACLK, upmode
            P1OUT ^= 0x03;
        }
        break;
    case END_GAME:
        if (count_diff > 0){
            UCA0TXBUF = black[j];
            j++;
```

```
            if (j == 25){
                j = 0;
                state = INIT; //get back to first stage in order to print new message
                UC0IE = 0; // make sure we will not get any interupts
                endgame = 1;
            }
            break;
        }
        else if (count_diff < 0){
            UCA0TXBUF = white[j];
            j++;
            if (j == 25){
                j = 0;
                state = INIT; //get back to first stage in order to print new message
                UC0IE = 0; // make sure we will not get any interupts
                endgame = 1;
            }
            break;
        }
        else {
            UCA0TXBUF = no_one[j];
            j++;
            if (j == 13){
                j = 0;
                state = INIT; //get back to first stage in order to print new message
                UC0IE = 0; // make sure we will not get any interupts
                endgame = 1;
            }
            break;
        }
    }
}
```