דוח מסכם - Design Review
פרוייקט לייזר טאג

1. <u>דרישות הלקוח:</u>
המשחק הוא "לייזר טאג" אבל עם תוספת תקשורת אלחוטית בין אקדח לשרת יעודי.
המערכת (המשחק) מורכבת מ 2 רכיבים: אקדח ושרת, ועל מנת לשחק דרושים לפחות 2 אקדחים.
על כל אקדח יש נורת אינפרא אדום וחיישן אינפרא אדום (אא), לחיצה על ההדק של האקדח תדליק את נורת ה-אא, וכאשר האקדח יכוון לגלאי ה-אא של האקדח שני, תירשם פגיעה באקדח ע"י נורית צבעונית ובנוסף האקדח הנפגע ישדר לשרת ע"מ לעקוב אחר התוצאה של המשחק.
תנאי המשחק נקבעים מראש ע"י השרת וניתן לשנותם בקוד, התנאים יכולים להיות מספר הפגיעות עד לסוף משחק או כמה זמן המשחק מתקיים.

2. <u>המערכת</u>:
המערכת מורכבת משני אלמנטים עיקריים: אקדח ושרת.

משדר ה-אא לא מחובר לבקר אלא לסוללה וכפתור חיצוניים מכיוון שמשדר ה-אא דורש מתח של V1.5 ולא רצינו לנסות להוציא מתח זה מהבקר

<u>רכיבי האקדח:</u>
- בקר 2274msp430f של חברת TI
- חיישן אינפרא אדום מסוג 1838VS
- נורת אינפרא אדום מסוג L-514EIR1C
- נוריות עם שלושה צבעים
- לחצן
- מתג
- חוטים\נגדים\בית סוללה\ ...

<u>רכיבי השרת:</u>
- בקר 2274msp430f של חברת TI
- הרחבת USB לבקר

האלמנטים מתקשרים בצורה אלחוטית ע"י רכיבי האלחוט של הבקר ובעזרת פרוטוקול SimpliciTI.
הבקרים שולחים אחד לשני הודעות בגודל שני בתים

<u>הודעת אקדח לשרת:</u>

score - 8-bit	**mode - 8-bit**
0 - 0xFF score	001 - ready 010 - score 0xFF - dead

- שדה mode מגדיר את סוג ההודעה
  - מצב ready משמש כהודעת ack
  - מצב score מודיע שהבית השני הוא תוצאת המשחק
  - מצב dead משמש כהודעת ack לפקודת kill

- שדה score שולח את מספר "הפגיעות" שספג האקדח
  - מספר בין 0 ל 255

הודעת שרת לאקדח:

mode - 8-bit	unused- 8-bit
001 - start 010 - hit_count 011 - reset 0xFF - kill	

- שדה mode מגדיר את סוג ההודעה
  - מצב start מודיע לאקדח על תחילת המשחק
  - מצב hit_count הוא בקשת שליחה של תוצאת המשחק העדכנית מהאקדח
  - מצב reset הוא פקודה לאקדח לאפס את מונה הפגיעות שלו ובעצם לאתחל את המשחק
  - מצב kill מודיע לאקדח שהמשחק נגמר

- השדה השני הוא לשימוש עתידי

חיבורים פיזיים:

מפרט חיבורים לרגליים של הבקר (פינים):

1. אדמה GND - רגל אמצעית של החיישן ורגל אדמה של הנורה
2. מתח VCC - רגל ימנית של החיישן
3. כניסה דיגיטלית - רגל שמאל של החיישן
4. יציאה דיגיטלית - רגל של נורה - אדום
5. יציאה דיגיטלית - רגל של נורה - ירוק
6. יציאה דיגיטלית - רגל של נורה - כחול

מהלך התקשורת בין השרת לאקדח:

S → G

mode - 8bit	unused - 8bit

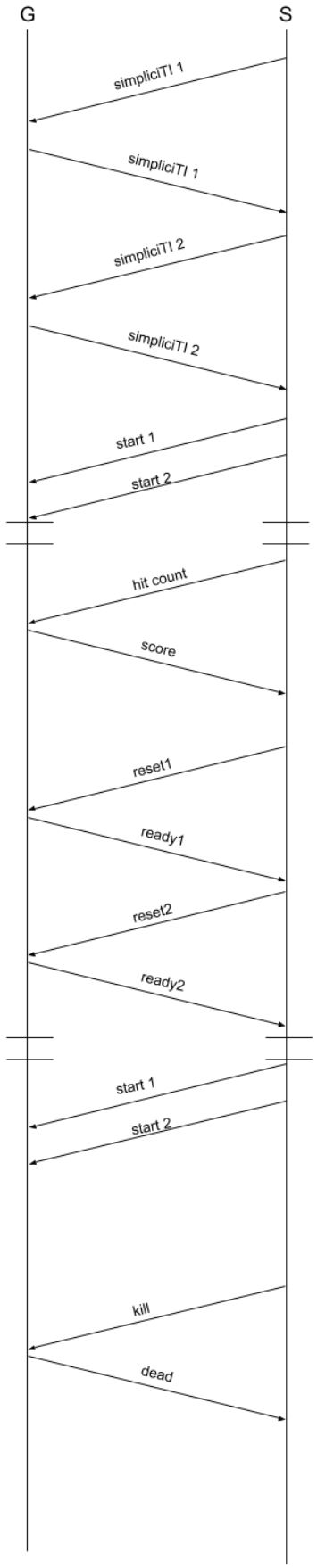
mode - 001 start
010 hit_count
011 reset
0xFF kill

G → S

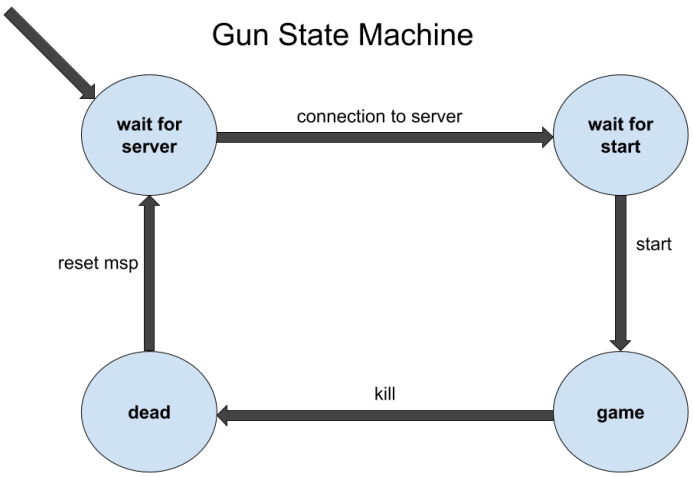
mode - 8bit	score- 8bit

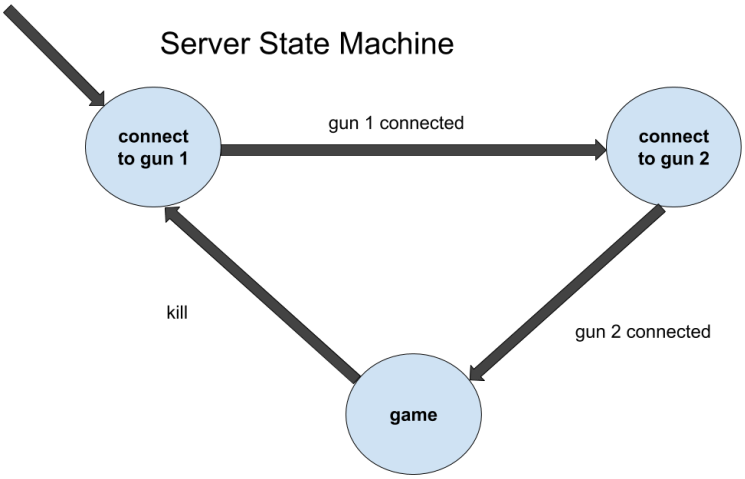
mode - 001 score
010 ready
0xFF dead

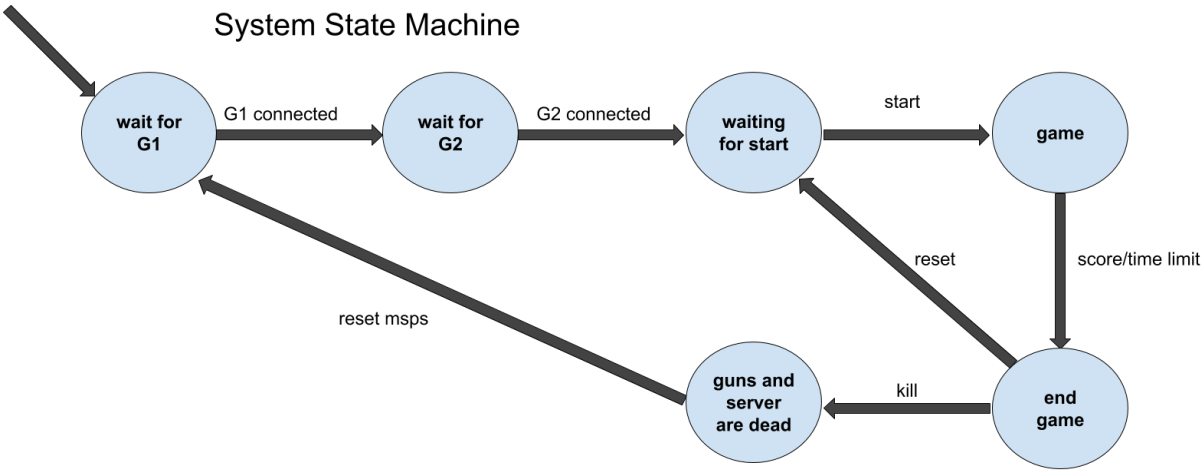
score - 0 - 0xFF score



מערכת המצבים של הפרוייקט:







מגבלות המערכת:

למערכת טווח פעילות מוגבל בשני אופנים שונים: תקשורת אלחוטית ואינפרא אדום.

- הטווח של התקשורת האלחוטית הוא כ 10 מטר (תלוי תנאי שטח).
- הטווח של האינפרא אדום הוא כ 15 מטר (תלוי תנאי שטח).

אנו מניחים שלא יתכן מצב של פסיקות מקוננות (מניחים שהפסיקות מטופלות במהירות ומבחינה הסתברותית לא תהיה התנגשות בפסיקות)

3. אלגוריתם:

במערכת אין אלגוריתם מוכר אך פרוטוקול מוכר שנעשה בו שימוש הוא פרוטוקול SmipliciTI של חברת Texas Instruments למימוש תקשורת אלחוטית בין בקרי msp.

שני האקדחים מתקשרים עם השרת באמצעות פרוטוקול זה.

4. <u>הפעלה:</u>

שלבי הפעלה

a. להרים את השרת מופיעות הודעות אודות המשחק והוראה כיצד להתחיל
   i. ש - שתי נורות דולקות מודפס Press the MSP button
b. לחיצה על כפתור השרת
   i. שתי הנורות מהבהבות ביחד מודפס "Search for GUNZZZZ to connect"
c. להדליק אקדח 1
   i. א - שתי נורות דולקות(פנימיות) + נורה (חיצונית) מהבהבת בכל הצבעים
d. אקדח 1 מתחבר
   i. א - נורות (פנימיות) אדום\ירוק מבהבהות ואז כבות.
   ii. ש - נורה ירוקה כבויה, נורה אדומה מהבהבת st player connected1
e. להדליק אקדח 2
   i. א - שתי נורות דולקות(פנימיות) + נורה (חיצונית) מהבהבת בכל הצבעים
f. אקדח 2 מתחבר
   i. א - נורות (פנימיות) אדום\ירוק מבהבהות ואז נכבות
   ii. ש - נורה אדומה כבויה nd player connected2
g. שני אקדחים מחוברים
   i. ש - שתי הנורות מהבהבות לסירוגין בקבלת הודעה.
h. שרת שולח start לשני האקדחים
   i. א - נורה ירוקה (חיצונית) מהבהבת והמשחק מתחיל
i. שרת שולח בקשות עדכון לאקדחים
   i. א - נורה (פנימית) ירוקה מהבהבת כל חצי שניה
j. הדפסה של התוצאות של האקדחים כאשר יש שינוי בתוצאה
k. אקדח שנפגע מדליק נורה אדומה (חיצונית ופנימית)
l. כשכמות הפגיעות מגיעה לסף שנקבע מראש המשחק נגמר
   i. אקדח מפסיד מהבהב את כל הצבעים
m. הדפסת תוצאת המשחק והכרזה על המנצח
   i. The winner is gun #_
n. הדפסת אפשרות להתחיל משחק חדש
   i. ש - would you like another game? press Y for yes, press N to finish
o. שליחת הודעת reset או kill בהתאם להחלטת השרת
   i. שליחת reset: א - נורה כחולה מהבהבת
   ii. שליחת reset: ש - ***Another Game Starts***
   iii. שליחת start: א - נורה ירוקה מהבהבת
   iv. שליחת kill: ש - Bye bye, thank you for playing
   v. שליחת kill: א - 4LPM
p. איתחול המשחק או כיבוי האקדחים

5. <u>קוד ופונקציות:</u>

<u>פונקציות אקדח:</u>

פונקציית **linkFrom**

פונקציה זו היא חלק מפרוטוקול SimpliciTI ותפקידה הוא לחבר את האקדח לשרת. כאשר האקדח התחבר הנורות יהבהבו, לאחר מכן הפונקציה מחכה לקלט אלחוטי מהשרת, ע"פ הפרוטוקול שכתבנו, על מנת לקבוע את הפעולה הבאה (האם לשלוח הודעת SOCRE/KILL/DEAD).
פונקציה זו לא מקבלת ערך ולא מחזירה ערך (void)
גודל הפונקציה בביתם ההוא 412

פונקציית **sRxCallback**

פונקציה זו היא חלק מפרוטוקול SimpliciTI ותפקידה לטפל בהודעות נכנסות. כאשר התקבלה הודעה הפונקציה בודקת אם היא מכתובת תקינה, אם כן, נבדוק את הבית השני של ההודעה ועל פיו נחליט איך להגיב (להחזיר SCORE או לבצע RESET או להתחיל את המשחק או לסיים את המשחק)..
פונקציה זו מקבלת את מספר הזהות של הרכיב שאיתו רוצים לתקשר ולא מחזירה כלום.
גודל הפונקציה בבית הוא 360

פונקציית **toggleLED**

פונקציה זו מדליקה את הנורה אם היא כבויה ומכבה אותה אם היא דולקת.
הפונקציה מקבלת בית אחד כדי לסמן איזה נורה להדליק/לכבות ולא מחזירה שום ערך.
גודל הפונקציה בבתים הוא 14

פונקציית **BSP_ISR_FUNCTION**

פונקציה זו מעדכנת את הבקר כך שפרוטוקול SimpliciTI לא יפריע לפסיקות שמגיעות ממקור חיצוני דרך פורט 2.

פונקציית **main**

פונקציה זו מקנפגת את הpin-ים החיצוניים (LED's, IR) לעבודה ב - Interrupts.
גודל הפונקציה בבתים הוא 236.

<u>פונקציות שרת:</u>

פונקציית **linkTo**:

פונקציה זו היא חלק מפרוטוקול SimpliciTI ותפקידה הוא לחבר בין האקדחים לשרת. כאשר ממתינים להתחברות האקדחים שתי הנורות מהבהבות יחדיו, ברגע שאקדח מתחבר מופיע פלט למסך המתאר התחברות, ונכבה הלד הירוק, בדומה קורה עבור האקדח הנוסף. לאחר מכן הפונקציה שולחת הודעת start לשני האקדחים והמשחק מתחיל. כעת בכל חצי שנייה נקבל אינטראפט מהטיימר ובכל פעם נשלח הודעה לסירוגין לכל אחד מהאקדחים שיחזיר לנו את התוצאה שלו, נפעל באופן זה עד הגעה לתוצאה שנקבעה, בכל שינוי בתוצאה היא מודפסת למסך.
בעת ההגעה לתוצאת ניצחון מכובה הטיימר ונבדק מנצח, מודפס למסך ונשלחת הודעה למסך האם רוצים משחק נוסף, במידה ונבחר 'כן' נשלחת לאקדחים הודעת reset אחרת הודעת kill.
עבור הודעת reset מאתחלים את המונים, מאתחלים את הטיימר ושולחים הודעת start.
עבור הודעת kill התוכנית נגמרת.
פונקציה זו לא מקבלת ערך ולא מחזירה ערך (void)
גודל הפונקציה בביתם ההוא 602

פונקציית **sRxCallback**

פונקציה זו היא חלק מפרוטוקול SimpliciTI ותפקידה לטפל בהודעות נכנסות. כאשר התקבלה הודעה הפונקציה בודקת אם היא מכתובת תקינה, מאיזה מבין שני האדקחים ההודעה התקבלה, ובהתאם נבדוק את הבית השני של ההודעה ועל פיו נחליט איך להגיב (להמשיך עם hit_count או לבדוק ready/dead).

פונקציה זו מקבלת את מספר הזהות של הרכיב שאיתו רוצים לתקשר ולא מחזירה כלום.

גודל הפונקציה בבית הוא 182

פונקציית **toggleLED**

פונקציה זו מדליקה את הנורה אם היא כבויה ומכבה אותה אם היא דולקת.

הפונקציה מקבלת בית אחד כדי לסמן איזה נורה להדליק/לכבות ולא מחזירה שום ערך.

גודל הפונקציה בבתים הוא 14

פונקציות**print_msg**

מדפיסות למסך מחרוזות.

גודל הפונקציה print_msg בבית הוא 28

פונקציית **Print_Score**

מדפיסה למסך את התוצאה העדכנית.

גודל הפונקציה בבית הוא 148

פונקציית **main**

פונקציה זו מאתחלת את הUART וממתינה ללחיצה על הכפתור לתחילת התוכנית. בנוסף היא מדפיסה למשתמש פרטים אודות המשחק.

גודל הפונקציה בבית הוא 210

6. <u>זכרון:</u>
קוד האקדח משתמש ב 8777 בתים ב FALSH
קוד האקדח משתמש ב 313 בתים ב RAM

קוד השרת משתמש ב 10339 בתים ב FALSH
קוד השרת משתמש ב 358 בתים ב RAM

7. <u>אופטימיזציה</u>:
<u>אופטימיזציה של אקדח:</u>
באופטימיזציה ברמת כל התוכנית, האקדח לא עבד בכלל (4O).
באופטימיזציה ברמת inter procedure האקדח מתפקד אבל ללא אא (3O).
באופטימיזציה ברמת global optimization האקדח מתפקד אבל ללא אא (2O).
באופטימיזציה ברמת local optimization האקדח מתפקד אבל ללא אא (1O).
האקדח מתפקד רק ברמת אופטימזציה 0O

<u>אופטימיזציה של שרת:</u>
השרת מתפקד ברמת אופטימיזציה של כל התוכנית (4O).

8. <u>צריכת זרם:</u>
צריכת זרם בשרת: השרת מחובר למחשב ולכן לא הרגשנו צורך לשנות את מצב חיסכון באנרגיה שלו (LPM) בשום שלב. צריכת הזרם היא 270 מיקרו אמפר
צריכת זרם באקדח: מאופי המשחק הלא צפוי קשה לשים מצב חיסכון באנרגיה, לכן רק כאשר נגמר המשחק הבקר עובר ל 4LPM. מכיוון שלבקר מחוברים חיישן ונורה קשה לנו לקבוע בוודאות את צריכת הזרם שלו, אך החיישן צורך 1.5 מילי אמפר בצריכה פסיבית.

9. <u>בעיות שנתקלנו בהן ואיך פתרנו אותן:</u>

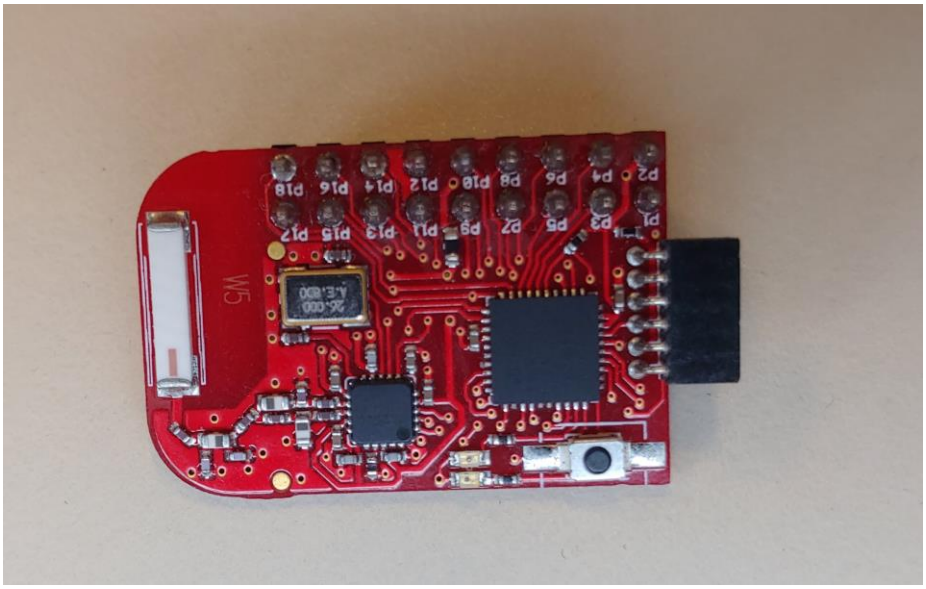
a. תחילה, לצורך הפרוייקט, התשמשנו בחיישן L-IR-T5B אך לאחר בדיקות תקינות גילינו שהחיישן לא מוציא מתח כנדרש (מתח נמוך מאוד).
דרכי הפעולה שחשבנו עליהן הן:
- לבנות מגבר לחיישן מטרניזטורים.
- להחליף חיישן.

דרך הפעולה שנבחרה היא להחליף חיישן. אפי דביר, ברוב טובו, נתן לנו חיישנים מסוג VS1838B שכבר מגבירים את האות בצורה מובנית.

b. ניסינו להפעיל את ה ADC על מנת לשפר את יכולת האבחון שלנו לפגיעה. לאחר כמה ימים מאוד מתסכלים וניסיונות של אדם לא הצלחנו להפעיל את ה ADC.
- פתרון: חיברנו את החיישן לכניסה דיגיטלית.

c. פרוטוקול SimpliciTI משתמש בשגרת פסיקות (ISR - Interrupt Service Routine) על 2port וגם חיישן ה-אא מחובר ל 2port עם פסיקות. חיבור של שני אלמנטים אלה ל 2port מייצר בעיה בקומפילציה של הקוד של הגדרה כפולה של שגרת הפסיקות.
- הפתרון היה למצוא את הקוד של ה SimpliciTI בבקר ולשים בהערה את חתיכת הקוד של שגרת הפסיקות ואז להוסיף אותה לקוד הראשי של התוכנית (בקובץ mrfi_board.c).

d. שניסינו לחבר שני בקרים לשרת אחד באמצעות SimpliciTI הם לא תקשרו רוב הזמן. אחרי דיבאג של כמה שעות גילינו שלבקרים יש מספר ליד האנטנה ואם לשני בקרים יש את אותו מספר זה יוצר בעיות.
- הפתרון להחליף לבקר עם מספר שונה (לא 5W).



בעיה נוספת היא וודו של הערות. לפעמים הקוד לא רץ כמו שצריך ואז אנחנו מוחקים הערה ואז הוא כן רץ.

10. <u>בדיקות תקינות:</u>
המערכת מתאוששת מאתחול של האקדחים. כמובן שאחרי אתחול האקדחים הם יעלו עם מונה פגיעות מאופס.
המערכת לא מתאוששת מאתחול של שרת.

11. <u>בדיקות **QA**:</u>
- בדיקות התבצעו בחלקים ועל קודי ניסוי.
- נכתב קוד ניסוי לבדיקת קבלת אותות אא.
- נכתב קוד ניסוי לבדיקת הנורות החיצוניות.
- בשלבים שונים של המשחק מופעלות הנורות הפנימיות של הבקר על מנת לשמש כ debug.
- ישנן בעיות של false positive בגילוי אא, נכון לעכשיו הבעיה ממש לא חמורה ולא מצריכה טיפול מיוחד. בממוצע על כל 15 פגיעות אמיתיות יש אחת false positive.
- רוב הצעדים במשחק מלווים בחיווי של נורות בצבעים שונים.

12. קוד:

קוד שרת:

```
#include "bsp.h"
#include "mrfi.h"
#include "nwk_types.h"
#include "nwk_api.h"
#include "bsp_leds.h"
#include "bsp_buttons.h"
#include "app_remap_led.h"
#define  END_SCORE 15
#define NUM_CONNECTIONS 2
extern int tempOffset; //code we added
static void linkTo(void);
void toggleLED(uint8_t);
void Print_Score();
void print_msg(char* str);

uint8_t gun1score=0;
uint8_t gun2score=0;
uint8_t gameOver=0;
int sendFlag = 1;//whom to send
int actionMode=1;//1-start,2-hit_count,3-reset,4-kill, 5-program flow
int readyCount=0;
char gun1win[] = {"The winner is gun #1\r\n"};
char gun2win[] = {"The winner is gun #2\r\n"};
static linkID_t sLinkIDs[NUM_CONNECTIONS] = {0};


/* application Rx frame handler. */
static uint8_t sRxCallback(linkID_t);

void main (void)
{
BSP_Init();

// LFXT1 = VLO  BCSCTL3 |= LFXT1S_2;

/*intial UART*/
// Set DCO       BCSCTL1 = CALBC1_1MHZ;
DCOCTL = CALDCO_1MHZ;
// P3.4,5 = USCI_A0 TXD/RXD         P3SEL |= 0x30;
// SMCLK          UCA0CTL1 = UCSSEL_2;
// 9600 from 1Mhz        UCA0BR0 = 104;
UCA0BR1 = 0;
UCA0MCTL = UCBRS_1;
// enable the RX        IE2  = 0x1;
```

```
_   __  __  ___ \r\n");     print_msg("
/_\\          | ____________________________________________________     print_msg("
|__ |__ |__/ \r\n");
|_  |___ /   \\ __||__ |  \\ \r\n");   /                                    print_msg("
|  \r\n"); ____________________                                  print_msg("   /
_|************************\r\n");   print_msg("  /
___\r\n");       ______   _          |     print_msg(" /
|    ____________________________________|          print_msg("/____
/_\\   |  __\r\n");
|   /   \\  |___|\r\n");          /                                    /     /          print_msg("
/----/  \r\n");       print_msg("   /
WELCOME TO THE BEST GAME  \r\n");          /        print_msg("  /
by\r\n");          /                    print_msg(" /
ASSAF & BARAK & TSLIL & HILA \r\n\n\n");     print_msg("/___________/


SMPL_Init(sRxCallback);

/* turn on LEDs. */
if (!BSP_LED2_IS_ON())
{
toggleLED(2);
}
if (!BSP_LED1_IS_ON())
{
toggleLED(1);
}
/* wait for a button press... */
print_msg("Press the MSP button\r\n");
do {
if (BSP_BUTTON1() || BSP_BUTTON2())
{
break;
}
} while (1);
print_msg("Searching for GUNZZZZ to connect\r\n");
/* never coming back... */
linkTo();
print_msg("\nBye Bye, Thank you for playing :)\r\n");
}

static void linkTo()
{
uint8_t  msg[2],numConnections=0;
int i;

while(SMPL_SUCCESS != SMPL_Link(&sLinkIDs[numConnections])) {
//SMPL_Link()
```

```
toggleLED(1);
toggleLED(2);
for(i=10000; i>0; i--);   // delay before sending
}
if (BSP_LED1_IS_ON())
{
toggleLED(1);
}
numConnections++;
print_msg("1st Player connected\r\n");
while(SMPL_SUCCESS != SMPL_Link(&sLinkIDs[numConnections])) {
//SMPL_Link()
toggleLED(2);
for(i=10000; i>0; i--);   // delay before sending
}
if (BSP_LED2_IS_ON())
{
toggleLED(2);
}
numConnections++;
print_msg("2nd Player connected\r\n");

// ACLK, upmode       TACTL = MC_0;

// TACCR0 interrupt enabled      TACCTL0 = CCIE;
// ~1/2 second        TACCR0 = 6000;
// ACLK, upmode        TACTL = TASSEL_1 + MC_1;
/* turn on RX. default is RX off. */
SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_RXON, 0);

print_msg("***Let The Games Begin***\r\n");
msg[0] = 1;  /* start msg */
msg[1]=END_SCORE;
SMPL_Send(sLinkIDs[0], msg, sizeof(msg));
SMPL_Send(sLinkIDs[1], msg, sizeof(msg));

actionMode=2;
msg[0] = 2;  /* hit count msg */
while (1)
{
__bis_SR_register(LPM0_bits + GIE);
if(actionMode==2){//hit count mode
if(gun1score<END_SCORE && gun2score<END_SCORE){//playing the game
msg[0] = 2;  /* hit count msg */
if(sendFlag==1){
SMPL_Send(sLinkIDs[0], msg, sizeof(msg));
sendFlag=2;
}
```

```
else{
SMPL_Send(sLinkIDs[1], msg, sizeof(msg));
sendFlag=1;
}
}
else{
// ACLK, upmode       TACTL = MC_0;
actionMode=5;
}
}
if(actionMode==5){//program flow mode
//checking who won
if(gun1score>=END_SCORE && gun2score<END_SCORE){
print_msg(gun1win);
Print_Score();
}
else if(gun2score>=END_SCORE && gun1score<END_SCORE){
print_msg(gun2win);
Print_Score();
}

print_msg("would you like another game?\r\n press Y for yes, press N to finish:  \r\n");

while(gameOver==0);
// print_msg("\r\n");
if(gameOver==1){
msg[0] = 3;  /* reset msg */
SMPL_Send(sLinkIDs[0], msg, sizeof(msg));
for(i=20000;i>0;i--);
SMPL_Send(sLinkIDs[1], msg, sizeof(msg));
for(i=20000;i>0;i--);
actionMode=3;
gameOver=0;
}
else if(gameOver==2){
msg[0] = 0xFF;  /* kill msg */
SMPL_Send(sLinkIDs[0], msg, sizeof(msg));
SMPL_Send(sLinkIDs[1], msg, sizeof(msg));
actionMode=4;
gameOver=0;
break;
}
}
//wite for 2 guns to be ready   if(actionMode==3){
//if (readyCount==2)

while(readyCount < 2);
readyCount=0;
```

```
msg[0] = 1;  /* start msg */
msg[1]=END_SCORE;
SMPL_Send(sLinkIDs[0], msg, sizeof(msg));
SMPL_Send(sLinkIDs[1], msg, sizeof(msg));
print_msg("\n***Another Game Starts***\r\n");
Print_Score();
actionMode=2;

// TACCR0 interrupt enabled      TACCTL0 = CCIE;
// ~1/2 second        TACCR0 = 6000;
// ACLK, upmode        TACTL = TASSEL_1 + MC_1;
}
if(actionMode==4){
while(readyCount < 2);//now waiting to 2 ready to die
break;
}
}
}


void toggleLED(uint8_t which)
{
if (1 == which)
{
BSP_TOGGLE_LED1();
}
else if (2 == which)
{
BSP_TOGGLE_LED2();
}
return;
}

/* handle received frames. */
static uint8_t sRxCallback(linkID_t port)
{
uint8_t recMsg[2], len, mode,score;
/* is the callback for the link ID we want to handle? */
if (port == sLinkIDs[0]) //receive message from gun 1
{
toggleLED(1);
/* yes. go get the frame. we know this call will succeed. */
if ((SMPL_SUCCESS == SMPL_Receive(sLinkIDs[0], recMsg, &len)) && len)
{
/* Check the application sequence number to detect
* late or missing frames...
*/
mode = recMsg[0];
```

```
score=recMsg[1];

if (mode==1)//update score
{
if(score!=gun2score){
toggleLED(1);        //
gun2score=score;
Print_Score();
}
}
else if(mode==2) //ready for new game - reset
{
gun2score=score;
readyCount++;
//set game timer??
}
else if(mode==0xFF) //ack to kill the game
{
readyCount++;
return 0;//kill??
}
/* drop frame. we're done with it. */
return 1;
}
}
else if (port == sLinkIDs[1])//receive message from gun 2
{
toggleLED(2);
/* yes. go get the frame. we know this call will succeed. */
if ((SMPL_SUCCESS == SMPL_Receive(sLinkIDs[1], recMsg, &len)) && len)
{

/* Check the application sequence number to detect
* late or missing frames...
*/
mode = recMsg[0];
score = recMsg[1];

if (mode==1)//update score
{
if(score!=gun1score){
// toggleLED(2);
gun1score=score;
Print_Score();
}
}
else if(mode==2) //ready for new game - reset
{
```

```
gun1score=score;
readyCount++;
//set game timer??
}
else if(mode==0xFF) //ack to kill the game
{
readyCount++;
return 0;//kill??
}
/* drop frame. we're done with it. */
return 1;
}
}
/* keep frame for later handling. */
return 0;
}

#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A (void)
{
// Clear CPUOFF bit from 0(SR)     __bic_SR_register_on_exit(CPUOFF);
}

#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCIAB0RX_ISR(void) {
char recievedChar;
if(IFG2 && UCA0RXBUF) {
// print the input chars into terminal      recievedChar = UCA0RXBUF;
UCA0TXBUF = UCA0RXBUF;

if(recievedChar=='Y'||recievedChar=='y'){
gameOver=1;
}
if(recievedChar=='N'||recievedChar=='n'){
gameOver=2;
}
else{
print_msg("wrong input try please try again\r\n");
}

}
LPM0_EXIT;
}



void print_msg(char* str) {
unsigned int j;
```

```
for (j = 0; str[j] != '\0'; j++) {
while(!(IFG2 & UCA0TXIFG)); //waiting for the buffer to be clear
UCA0TXBUF = str[j];

}
}

void Print_Score(){
char sNum[6]={0,0,0,0,0,'\0'};
if(gun1score>=END_SCORE){
gun1score=END_SCORE;
}
if(gun2score>=END_SCORE){
gun2score=END_SCORE;
}
sNum[0]=(int)gun1score/10+48;
sNum[1]=(int)gun1score%10+48;
sNum[2]=':';
sNum[3]=(int)gun2score/10+48;
sNum[4]=(int)gun2score%10+48;
print_msg("The game score is: ");
print_msg(sNum);
print_msg("\r\n");
}
```

קוד אקדח:

```
/*-----------------------------------------------------------------------------
 *  Demo Application for SimpliciTI
 *
 *  L. Friedman
 *  Texas Instruments, Inc.
 *-----------------------------------------------------------------------------
 */
/*****************************************************************************************
```

```
#include "nwk_types.h"
#include "nwk_api.h"
#include "bsp_leds.h"
#include "bsp_buttons.h"
#include <msp430.h>
#include "app_remap_led.h"

#include "mrfi_defs.h"







static void linkFrom(void);

void toggleLED(uint8_t);

void print(char msg[]);

void print_counter(void);


uint8_t  sRxTid = 0;       //static
linkID_t sLinkID2 = 0;         static
static volatile uint8_t  sSemaphore = 0;
~~~~~~~~~~~~~~~~~*/   OUR GLOBAL VARIABLES        /*~~~~~~~~~~~~~


uint8_t mode=0;
uint8_t maxHits,hit_counter=0;
int ii,z,index=0;//,floop;
/* ADC */
int degC, irFlag=0,delay;
volatile long temp;
int results[0];
extern int tempOffset;

/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/



/* Rx callback handler */
static uint8_t sRxCallback(linkID_t);

void main (void)
{
```

```
BSP_Init();

/* If an on-the-fly device address is generated it must be done before the
* call to SMPL_Init(). If the address is set here the ROM value will not
* be used. If SMPL_Init() runs before this IOCTL is used the IOCTL call
* will not take effect. One shot only. The IOCTL call below is conformal.
*/
#ifdef I_WANT_TO_CHANGE_DEFAULT_ROM_DEVICE_ADDRESS_PSEUDO_CODE
{
addr_t lAddr;

createRandomAddress(&lAddr);
SMPL_Ioctl(IOCTL_OBJ_ADDR, IOCTL_ACT_SET, &lAddr);
}
#endif /* I_WANT_TO_CHANGE_DEFAULT_ROM_DEVICE_ADDRESS_PSEUDO_CODE
*/

/* This call will fail because the join will fail since there is no Access Point
* in this scenario. But we don't care -- just use the default link token later.
* We supply a callback pointer to handle the message returned by the peer.
*/
SMPL_Init(sRxCallback);



/* turn on LEDs. */
if (!BSP_LED2_IS_ON())
{
toggleLED(2);
}
if (!BSP_LED1_IS_ON())
{
toggleLED(1);
}

/* wait for a button press... */          //
do {          //
if (BSP_BUTTON1() || BSP_BUTTON2())      //
{      //
break;  //
}      //
} while (1);          //


/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
// stop watchdog timer  WDTCTL = WDTPW | WDTHOLD;
```

```
// configure P1.0 as output        P1DIR |= 0x01;

// make pin P1.2 input (ECHO)  P2DIR &= ~0x01;
P2SEL &= 0x0;
// enable interupt on ECHO pin      P2IE |= 0x01;
// make pin P2.1,P2.2,P2.3 output (External LED's)     P2DIR |= 0x0E;

/* Timer A */
// timer A config    TACCTL0 = CCIE;
TACCR0 = 6000;


*/       /*  UART Interrupt Configuration
// setting the DCOCLK to 1 MHz         BCSCTL1 = CALBC1_1MHZ;
// setting the DCOCLK to 1 MHz         DCOCTL = CALDCO_1MHZ;

//  "Reset" Flag interrupt Rx & Tx           IFG2 &= 0xFC;
// Interrupt Rx & Tx Enable         IE2 |= UCA0RXIE;
for(z=4;z>0;z--)
{
// toggle P1.0          P2OUT ^= 0x02;
// toggle P1.0          P2OUT ^= 0x02;
// toggle P1.0          P2OUT ^= 0x02;
// delay          for(ii=10000; ii>0; ii--);
// toggle P1.0          P2OUT ^= 0x04;
// toggle P1.0          P2OUT ^= 0x04;
// toggle P1.0          P2OUT ^= 0x04;
// delay          for(ii=10000; ii>0; ii--);
// toggle P1.0          P2OUT ^= 0x08;
// toggle P1.0          P2OUT ^= 0x08;
// toggle P1.0          P2OUT ^= 0x08;
// delay          for(ii=10000; ii>0; ii--);
}
P2OUT &= 0xF1;
__bis_SR_register(GIE);
print("Temp: ");         //

/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
/* never coming back... */
linkFrom();

/* but in case we do... */
while (1) ;
}

static void linkFrom()
{
```

```
msg[2];//, tid = 0;           uint8_t

/* Turn off one LED so we can tell the device is now listening.
* Received messages will toggle the other LED.
*/
toggleLED(1);//Green LED off

/* listen for link forever... */
while (1)
{
if (SMPL_SUCCESS == SMPL_LinkListen(&sLinkID2))
{
break;
}
/* Implement fail-to-link policy here. otherwise, listen again. */
}
toggleLED(1);
toggleLED(2);
for(delay=10000; delay>0; delay--);
toggleLED(1);
toggleLED(2);
for(delay=10000; delay>0; delay--);
/* turn off LEDs. */
if (BSP_LED2_IS_ON())
{
toggleLED(2);
}
if (BSP_LED1_IS_ON())
{
toggleLED(1);
}

/* turn on LED1 on the peer in response to receiving a frame. */
*msg = 0x01;          //

/* turn on RX. default is RX off. */
SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_RXON, 0);

while (1)
{
/* Wait for a frame to be received. The Rx handler, which is running in
* ISR thread, will post to this semaphore allowing the application to
* send the reply message in the user thread.
*/
if (sSemaphore)
{
if(mode == 2)//server wants hit_counter update -> send "SCORE"
{
```

```
toggleLED(1);
*(msg+1) = hit_counter;
*msg = 0x01;
}
else if(mode == 3)//server sent "RESET" -> send "READY"
{
*msg = 0x02;
*(msg+1) = 0;
}
else if(mode == 0xFF)//GOT "KILL" -> SEND "DEAD"
{
*msg = 0xFF;
*(msg+1) = 0xFF;
}

*(msg+1) = ++tid;  //
SMPL_Send(sLinkID2, msg, 2);

/* Reset semaphore. This is not properly protected and there is a race
* here. In theory we could miss a message. Good enough for a demo, though.
*/
sSemaphore = 0;
if(mode == 0xFF){ _BIS_SR(LPM4_bits); }
}
if ((P2IFG & 0x01) && (irFlag == 1))
{
hit_counter++;
if(hit_counter > maxHits)
{
for(z=4;z>0;z--)
{
// toggle P1.0          P2OUT ^= 0x02;
// toggle P1.0          P2OUT ^= 0x02;
// toggle P1.0          P2OUT ^= 0x02;
// delay          for(ii=10000; ii>0; ii--);
// toggle P1.0          P2OUT ^= 0x04;
// toggle P1.0          P2OUT ^= 0x04;
// toggle P1.0          P2OUT ^= 0x04;
// delay          for(ii=10000; ii>0; ii--);
// toggle P1.0          P2OUT ^= 0x08;
// toggle P1.0          P2OUT ^= 0x08;
// toggle P1.0          P2OUT ^= 0x08;
// delay          for(ii=10000; ii>0; ii--);
P2OUT &= 0xF1;
}
hit_counter = hit_counter;
}
P2OUT &= 0xFD;
```

```
// toggle P1.0          P2OUT ^= 0x02;
// delay          for(ii=20000; ii>0; ii--);
P2OUT &= 0xFD;
// toggle P1.0          P1OUT ^= 0x01;
P2IFG &= 0;
irFlag = 0;  //
// choose clock (ACLK) + UP mode       TACTL = TASSEL_1 + MC_1;      //
}
}
}

void toggleLED(uint8_t which)
{
if (1 == which)
{
BSP_TOGGLE_LED1();
}
else if (2 == which)
{
BSP_TOGGLE_LED2();
}
return;
}

/* handle received messages */
static uint8_t sRxCallback(linkID_t port)
{
uint8_t msg[2], len;


/* is the callback for the link ID we want to handle? */
if (port == sLinkID2)
{
/* yes. go get the frame. we know this call will succeed. */
if ((SMPL_SUCCESS == SMPL_Receive(sLinkID2, msg, &len)) && len)
{
mode = msg[0];
if(mode == 2)//"HIT_COUNT" - server wants hit_counter update
{
/* Post to the semaphore to let application know so it sends the reply */
sSemaphore = 1;
}
else if(mode == 3)//"RESET" - we have a winner but they want to play again
{
P2OUT |= 0x08;
hit_counter = 0;
irFlag = 0;
sSemaphore = 1;
```

```
}
else if(mode == 1)//"START" - communication established, let's start play!
{
maxHits = msg[1];
irFlag = 1;/* enable IR interrupt enable
*///~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
P2OUT &= 0xF1;
toggleLED(1);
// toggle P1.0         P2OUT ^= 0x04;
for(delay=10000; delay>0; delay--);
toggleLED(1);
// toggle P1.0         P2OUT ^= 0x04;
for(delay=10000; delay>0; delay--);
toggleLED(1);
// toggle P1.0         P2OUT ^= 0x04;
for(delay=10000; delay>0; delay--);
toggleLED(1);
// toggle P1.0         P2OUT ^= 0x04;
for(delay=10000; delay>0; delay--);
toggleLED(2);
// toggle P1.0         P2OUT ^= 0x04;
for(delay=10000; delay>0; delay--);
toggleLED(2);
// toggle P1.0         P2OUT ^= 0x04;
for(delay=10000; delay>0; delay--);
toggleLED(2);
// toggle P1.0         P2OUT ^= 0x04;
for(delay=10000; delay>0; delay--);
toggleLED(2);
// toggle P1.0        P2OUT &= 0xFB;
for(delay=10000; delay>0; delay--);
}
else if(mode == 0xFF)//"KILL" - "GameOver" im done.
{
sSemaphore = 1;
}
else{}//for future purposes

/* drop frame. we're done with it. */
return 1;
}
}
/* keep frame for later handling */
return 0;
}

// interrupt "handler"         #pragma vector = TIMERA0_VECTOR
__interrupt void Timer_A (void)
```

```
{
// toggle P1.0          P1OUT ^= 0x01;                    //          //
irFlag = 1;          //
// choose clock (ACLK) + STOP mode       TACTL = TASSEL_1 + MC_0;              //
}

BSP_ISR_FUNCTION( BSP_GpioPort1Isr, PORT2_VECTOR )
{
if(P2IFG & 0x01)
{
// toggle P1.0          P1OUT ^= 0x01;                //
P2IFG &= 0;
}


MRFI_GpioIsr();
}
```