דוח מסכם - Design Review

פרוייקט לייזר טאג



The A-Team:

הילה גורביץ' 2044340996

צליל גרינברג 313347841

אסף עומסי 201656063

ברק פלד 302788013

1. **דרישות הלקוח:**

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.

1. **המערכת**:

המערכת מורכבת משני אלמנטים עיקריים: אקדח ושרת.

משדר ה-אא לא מחובר לבקר אלא לסוללה וכפתור חיצוניים מכיוון שמשדר ה-אא דורש מתח של 1.5V ולא רצינו לנסות להוציא מתח זה מהבקר

רכיבי האקדח:

* בקר msp430f2274 של חברת TI
* חיישן אינפרא אדום מסוג VS1838
* נורת אינפרא אדום מסוג L-514EIR1C
* נוריות עם שלושה צבעים
* לחצן
* מתג
* חוטים\נגדים\בית סוללה\ ...

רכיבי השרת:

* בקר msp430f2274 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

הודעת אקדח לשרת:

|  |  |
| --- | --- |
| **score - 8-bit**  0 - 0xFF score | **mode - 8-bit**  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:

|  |  |
| --- | --- |
| **unused- 8-bit** | **mode - 8-bit**  001 - start  010 - hit\_count  011 - reset  0xFF - kill |

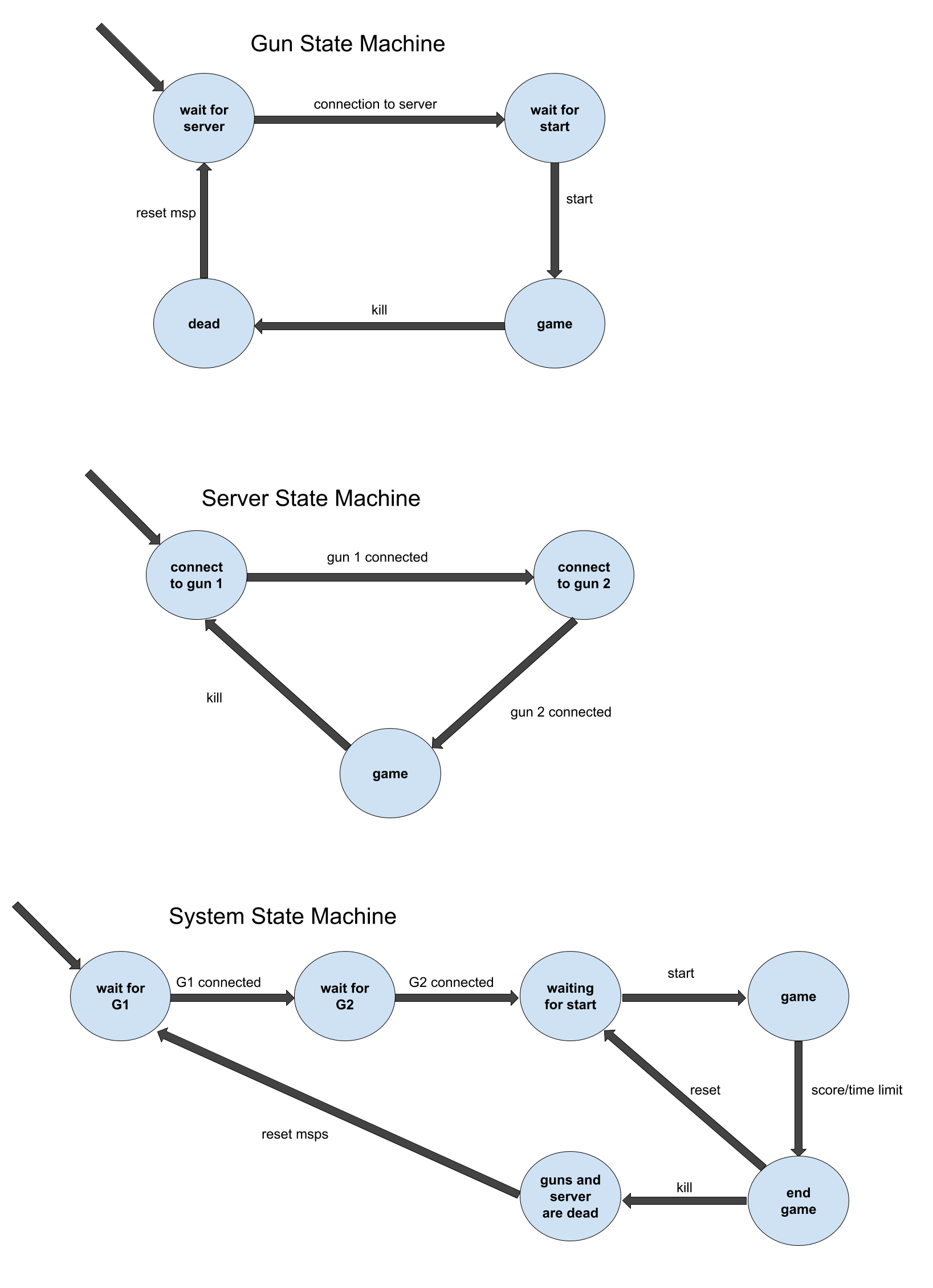
* שדה mode מגדיר את סוג ההודעה
  + מצב start מודיע לאקדח על תחילת המשחק
  + מצב hit\_count הוא בקשת שליחה של תוצאת המשחק העדכנית מהאקדח
  + מצב reset הוא פקודה לאקדח לאפס את מונה הפגיעות שלו ובעצם לאתחל את המשחק
  + מצב kill מודיע לאקדח שהמשחק נגמר
* השדה השני הוא לשימוש עתידי

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:

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)

1. **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.

1. **הפעלה:**

שלבי הפעלה

* 1. להרים את השרת מופיעות הודעות אודות המשחק והוראה כיצד להתחיל
     1. ש - שתי נורות דולקות מודפס Press the MSP button
  2. לחיצה על כפתור השרת
     1. שתי הנורות מהבהבות ביחד מודפס "Search for GUNZZZZ to connect"
  3. להדליק אקדח 1
     1. א - שתי נורות דולקות(פנימיות) + נורה (חיצונית) מהבהבת בכל הצבעים
  4. אקדח 1 מתחבר
     1. א - נורות (פנימיות) אדום\ירוק מבהבהות ואז כבות.
     2. ש - נורה ירוקה כבויה, נורה אדומה מהבהבת 1st player connected
  5. להדליק אקדח 2
     1. א - שתי נורות דולקות(פנימיות) + נורה (חיצונית) מהבהבת בכל הצבעים
  6. אקדח 2 מתחבר
     1. א - נורות (פנימיות) אדום\ירוק מבהבהות ואז נכבות
     2. ש - נורה אדומה כבויה 2nd player connected
  7. שני אקדחים מחוברים
     1. ש - שתי הנורות מהבהבות לסירוגין בקבלת הודעה.
  8. שרת שולח start לשני האקדחים
     1. א - נורה ירוקה (חיצונית) מהבהבת והמשחק מתחיל
  9. שרת שולח בקשות עדכון לאקדחים
     1. א - נורה (פנימית) ירוקה מהבהבת כל חצי שניה
  10. הדפסה של התוצאות של האקדחים כאשר יש שינוי בתוצאה
  11. אקדח שנפגע מדליק נורה אדומה (חיצונית ופנימית)
  12. כשכמות הפגיעות מגיעה לסף שנקבע מראש המשחק נגמר
      1. אקדח מפסיד מהבהב את כל הצבעים
  13. הדפסת תוצאת המשחק והכרזה על המנצח
      1. The winner is gun #\_
  14. הדפסת אפשרות להתחיל משחק חדש
      1. ש - would you like another game? press Y for yes, press N to finish
  15. שליחת הודעת reset או kill בהתאם להחלטת השרת
      1. שליחת reset: א - נורה כחולה מהבהבת
      2. שליחת :reset ש - \*\*\*Another Game Starts\*\*\*
      3. שליחת start: א - נורה ירוקה מהבהבת
      4. שליחת kill: ש - Bye bye, thank you for playing
      5. שליחת kill: א - LPM4
  16. איתחול המשחק או כיבוי האקדחים

1. **קוד ופונקציות:**

פונקציות אקדח:

**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.

פונקציות שרת:

**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

1. **זכרון:**

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

1. **אופטימיזציה**:

אופטימיזציה של אקדח:

באופטימיזציה ברמת כל התוכנית, האקדח לא עבד בכלל (O4).

באופטימיזציה ברמת inter procedure האקדח מתפקד אבל ללא אא (O3).

באופטימיזציה ברמת global optimization האקדח מתפקד אבל ללא אא (O2).

באופטימיזציה ברמת local optimization האקדח מתפקד אבל ללא אא (O1).

האקדח מתפקד רק ברמת אופטימזציה O0

אופטימיזציה של שרת:

השרת מתפקד ברמת אופטימיזציה של כל התוכנית (O4).

1. **צריכת זרם:**

**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

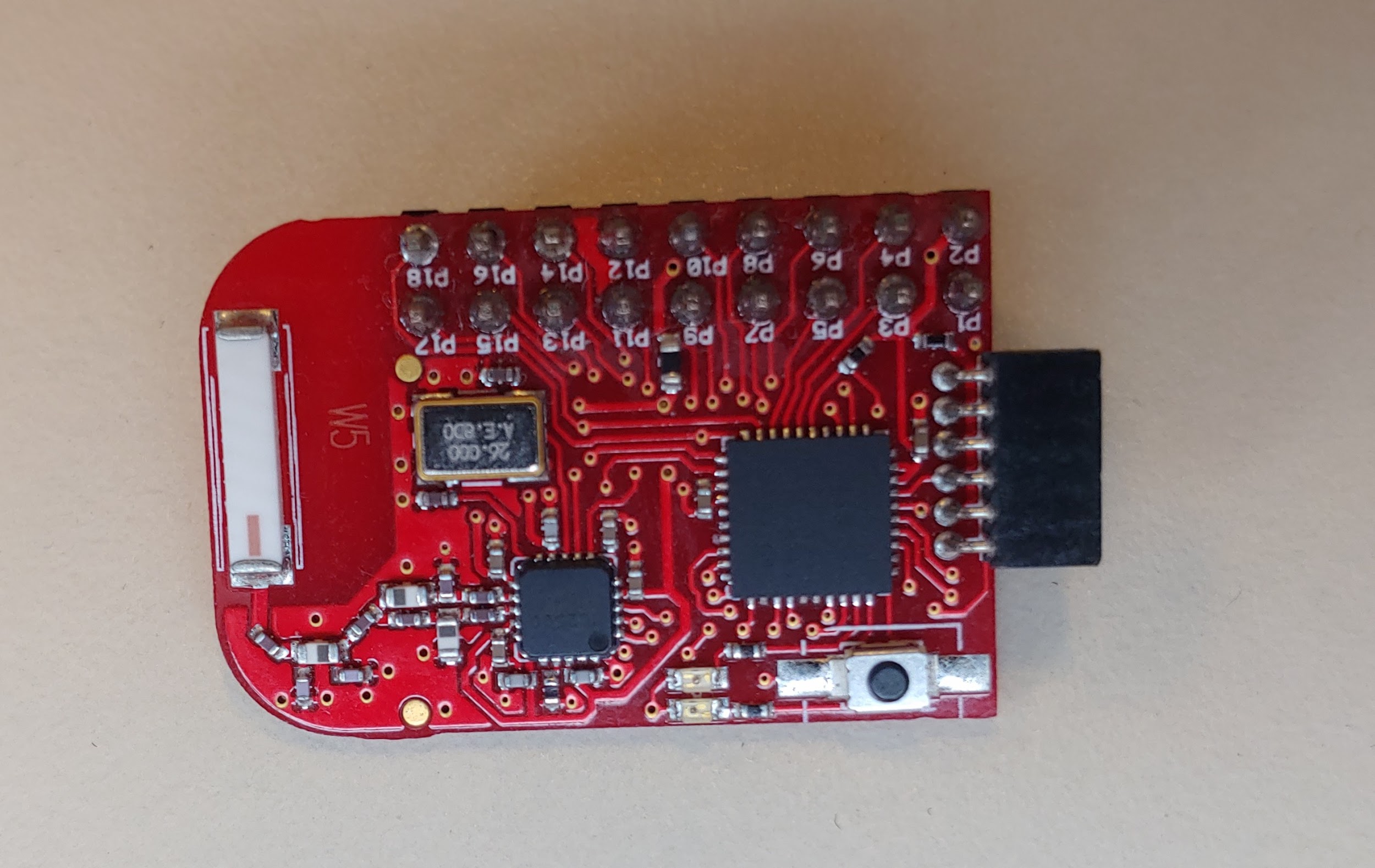
**צריכת זרם באקדח**: מאופי המשחק הלא צפוי קשה לשים מצב חיסכון באנרגיה, לכן רק כאשר נגמר המשחק הבקר עובר ל LPM4. מכיוון שלבקר מחוברים חיישן ונורה קשה לנו לקבוע בוודאות את צריכת הזרם שלו, אך החיישן צורך 1.5 מילי אמפר בצריכה פסיבית.

1. **בעיות שנתקלנו בהן ואיך פתרנו אותן:**
   1. 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.

* 1. 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.
  1. פרוטוקול SimpliciTI משתמש בשגרת פסיקות (ISR - Interrupt Service Routine) על port2 וגם חיישן ה-אא מחובר ל port2 עם פסיקות. חיבור של שני אלמנטים אלה ל port2 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).
  1. 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 W5).



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

1. **בדיקות תקינות:**

המערכת מתאוששת מאתחול של האקדחים. כמובן שאחרי אתחול האקדחים הם יעלו עם מונה פגיעות מאופס.

המערכת לא מתאוששת מאתחול של שרת.

1. **בדיקות QA:**

* 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.

1. **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();

BCSCTL3 |= LFXT1S\_2; // LFXT1 = VLO

/\*intial UART\*/

BCSCTL1 = CALBC1\_1MHZ; // Set DCO

DCOCTL = CALDCO\_1MHZ;

P3SEL |= 0x30; // P3.4,5 = USCI\_A0 TXD/RXD

UCA0CTL1 = UCSSEL\_2; // SMCLK

UCA0BR0 = 104; // 9600 from 1Mhz

UCA0BR1 = 0;

UCA0MCTL = UCBRS\_1;

IE2 = 0x1; // enable the RX

print\_msg(" \_ \_\_ \_\_ \_\_\_ \r\n");

print\_msg(" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | /\_\\ |\_\_ |\_\_ |\_\_/ \r\n");

print\_msg(" / |\_ |\_\_\_ / \\ \_\_||\_\_ | \\ \r\n");

print\_msg(" / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | \r\n");

print\_msg(" / \_|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\r\n");

print\_msg(" / | \_\_\_\_\_ \_ \_\_\_\r\n");

print\_msg("/\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_| | /\_\\ | \_\_\r\n");

print\_msg(" / / / | / \\ |\_\_\_|\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");

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");

TACTL = MC\_0; // ACLK, upmode

TACCTL0 = CCIE; // TACCR0 interrupt enabled

TACCR0 = 6000; // ~1/2 second

TACTL = TASSEL\_1 + MC\_1; // ACLK, upmode

/\* 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{

TACTL = MC\_0; // ACLK, upmode

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;

}

}

if(actionMode==3){ //wite for 2 guns to be ready

//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;

TACCTL0 = CCIE; // TACCR0 interrupt enabled

TACCR0 = 6000; // ~1/2 second

TACTL = TASSEL\_1 + MC\_1; // ACLK, upmode

}

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)

{

\_\_bic\_SR\_register\_on\_exit(CPUOFF); // Clear CPUOFF bit from 0(SR)

}

#pragma vector=USCIAB0RX\_VECTOR

\_\_interrupt void USCIAB0RX\_ISR(void) {

char recievedChar;

if(IFG2 && UCA0RXBUF) {

recievedChar = UCA0RXBUF; // print the input chars into terminal

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.

\*----------------------------------------------------------------------------

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Copyright 2007-2009 Texas Instruments Incorporated. All rights reserved.

IMPORTANT: Your use of this Software is limited to those specific rights granted under

the terms of a software license agreement between the user who downloaded the software,

his/her employer (which must be your employer) and Texas Instruments Incorporated (the

"License"). You may not use this Software unless you agree to abide by the terms of the

License. The License limits your use, and you acknowledge, that the Software may not be

modified, copied or distributed unless embedded on a Texas Instruments microcontroller

or used solely and exclusively in conjunction with a Texas Instruments radio frequency

transceiver, which is integrated into your product. Other than for the foregoing purpose,

you may not use, reproduce, copy, prepare derivative works of, modify, distribute,

perform, display or sell this Software and/or its documentation for any purpose.

YOU FURTHER ACKNOWLEDGE AND AGREE THAT THE SOFTWARE AND DOCUMENTATION ARE PROVIDED “AS IS”

WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY

WARRANTY OF MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.

IN NO EVENT SHALL TEXAS INSTRUMENTS OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER CONTRACT,

NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR OTHER LEGAL EQUITABLE

THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES INCLUDING BUT NOT LIMITED TO ANY

INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST

DATA, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY

THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.

Should you have any questions regarding your right to use this Software,

contact Texas Instruments Incorporated at www.TI.com.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "bsp.h"

#include "mrfi.h"

#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);

//static uint8\_t sRxTid = 0;

static linkID\_t sLinkID2 = 0;

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);

/\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*/

WDTCTL = WDTPW | WDTHOLD; // stop watchdog timer

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

P2DIR &= ~0x01; // make pin P1.2 input (ECHO)

P2SEL &= 0x0;

P2IE |= 0x01; // enable interupt on ECHO pin

P2DIR |= 0x0E; // make pin P2.1,P2.2,P2.3 output (External LED's)

/\* Timer A \*/

TACCTL0 = CCIE; // timer A config

TACCR0 = 6000;

/\* UART Interrupt Configuration \*/

BCSCTL1 = CALBC1\_1MHZ; // setting the DCOCLK to 1 MHz

DCOCTL = CALDCO\_1MHZ; // setting the DCOCLK to 1 MHz

IFG2 &= 0xFC; // "Reset" Flag interrupt Rx & Tx

IE2 |= UCA0RXIE; // Interrupt Rx & Tx Enable

for(z=4;z>0;z--)

{

P2OUT ^= 0x02; // toggle P1.0

P2OUT ^= 0x02; // toggle P1.0

P2OUT ^= 0x02; // toggle P1.0

for(ii=10000; ii>0; ii--); // delay

P2OUT ^= 0x04; // toggle P1.0

P2OUT ^= 0x04; // toggle P1.0

P2OUT ^= 0x04; // toggle P1.0

for(ii=10000; ii>0; ii--); // delay

P2OUT ^= 0x08; // toggle P1.0

P2OUT ^= 0x08; // toggle P1.0

P2OUT ^= 0x08; // toggle P1.0

for(ii=10000; ii>0; ii--); // delay

}

P2OUT &= 0xF1;

\_\_bis\_SR\_register(GIE);

// print("Temp: ");

/\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*/

/\* never coming back... \*/

linkFrom();

/\* but in case we do... \*/

while (1) ;

}

static void linkFrom()

{

uint8\_t msg[2];//, tid = 0;

/\* 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--)

{

P2OUT ^= 0x02; // toggle P1.0

P2OUT ^= 0x02; // toggle P1.0

P2OUT ^= 0x02; // toggle P1.0

for(ii=10000; ii>0; ii--); // delay

P2OUT ^= 0x04; // toggle P1.0

P2OUT ^= 0x04; // toggle P1.0

P2OUT ^= 0x04; // toggle P1.0

for(ii=10000; ii>0; ii--); // delay

P2OUT ^= 0x08; // toggle P1.0

P2OUT ^= 0x08; // toggle P1.0

P2OUT ^= 0x08; // toggle P1.0

for(ii=10000; ii>0; ii--); // delay

P2OUT &= 0xF1;

}

hit\_counter = hit\_counter;

}

P2OUT &= 0xFD;

P2OUT ^= 0x02; // toggle P1.0

for(ii=20000; ii>0; ii--); // delay

P2OUT &= 0xFD;

P1OUT ^= 0x01; // toggle P1.0

P2IFG &= 0;

// irFlag = 0;

// TACTL = TASSEL\_1 + MC\_1; // choose clock (ACLK) + UP mode

}

}

}

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);

P2OUT ^= 0x04; // toggle P1.0

for(delay=10000; delay>0; delay--);

toggleLED(1);

P2OUT ^= 0x04; // toggle P1.0

for(delay=10000; delay>0; delay--);

toggleLED(1);

P2OUT ^= 0x04; // toggle P1.0

for(delay=10000; delay>0; delay--);

toggleLED(1);

P2OUT ^= 0x04; // toggle P1.0

for(delay=10000; delay>0; delay--);

toggleLED(2);

P2OUT ^= 0x04; // toggle P1.0

for(delay=10000; delay>0; delay--);

toggleLED(2);

P2OUT ^= 0x04; // toggle P1.0

for(delay=10000; delay>0; delay--);

toggleLED(2);

P2OUT ^= 0x04; // toggle P1.0

for(delay=10000; delay>0; delay--);

toggleLED(2);

P2OUT &= 0xFB; // toggle P1.0

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;

}

#pragma vector = TIMERA0\_VECTOR // interrupt "handler"

\_\_interrupt void Timer\_A (void)

{

// // P1OUT ^= 0x01; // toggle P1.0

// irFlag = 1;

// TACTL = TASSEL\_1 + MC\_0; // choose clock (ACLK) + STOP mode

}

BSP\_ISR\_FUNCTION( BSP\_GpioPort1Isr, PORT2\_VECTOR )

{

if(P2IFG & 0x01)

{

// P1OUT ^= 0x01; // toggle P1.0

P2IFG &= 0;

}

MRFI\_GpioIsr();

}