Răzvan - Gabriel PREPELIUC

PROIECT DE DIPLOMĂ

SMART DOG COLLAR

SPECIALIZAREA ELECTRONICĂ APLICATĂ

Marchează dacă ai studiat în limba engleză: ÎN LIMBA ENGLEZĂ

☑

FACULTATEA DE ELECTRONICĂ, TELECOMUNICAȚII
ȘI TEHNOLOGIA INFORMAȚIEI
UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

2021

Str. George Barițiu, nr. 26-28, 400027, Cluj-Napoca, tel: 0264-401224, tel/fax: 0264-591689, etti.utcluj.ro

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA
FACULTATEA DE ELECTRONICĂ
TELECOMUNICAȚII ȘI TEHNOLOGIA INFORMAȚIEI
Specializarea: Electronică Aplicată (în limba engleză)

SMART DOG COLLAR

Proiect de diplomă

Absolvent,
Răzvan-Gabriel PREPELIUC

Razvan Prepeliuc

Decan,
Prof.dr.ing. Gabriel OLTEAN

Președinte comisie,
Prof.dr.ing. Dan PITICĂ

2021

Această pagină a fost lăsată goală în mod intenționat.

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA
FACULTATEA DE ELECTRONICĂ
TELECOMUNICAȚII ȘI TEHNOLOGIA INFORMAȚIEI

Departamentul Electronică Aplicată

Titlul proiectului de diplomă:

SMART DOG COLLAR

Descrierea temei:

This project presents the concept of a Smart Dog Collar using the MSP432P401R LaunchPad from Texas Instruments. I plan on implementing more features to not only make the life of a dog better, but also the life of the owner.
The project consists of three main parts: an alarm based on two LEDs and a piezo-buzzer that will trigger when an impact is detected by a 3-axis accelerometer and is stopped by pressing a push-button, a GPS module that will display the latitude and longitude continuously for the owner to know the position of his dog and an RFID module used to keep count of the times a dog has gone to the food bowl.

Locul de realizare:
Universitatea Tehnică din Cluj Napoca

Adaugă consultant: ☐

Data emiterii temei: 02.11.2020

Data predării temei: 09.07.2021

Absolvent,

Răzvan-Gabriel PREPELIUC

Razvan Prepeliuc

Director departament,

Prof.dr.ing. Dorin PETREUȘ

Conducător,

Ș.l.dr.ing. Mihai DĂRĂBAN

Această pagină a fost lăsată goală în mod intenționat.

Declaraţie pe proprie răspundere privind autenticitatea lucrării de diplomă

Subsemnatul **Răzvan-Gabriel PREPELIUC**
legitimat cu CI seria XV nr. 446983 CNP 1980731226717
în calitate de autor al lucrării cu titlul

SMART DOG COLLAR

elaborată în vederea susţinerii examenului de finalizare a studiilor de licenţă, la
Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei
Specializarea Electronică Aplicată (în limba engleză)
Universitatea Tehnică din Cluj-Napoca
sesiunea iulie a anului 2021

declar pe proprie răspundere că această lucrare este rezultatul propriei activităţi intelectuale, pe baza cercetărilor mele şi pe baza informaţiilor obţinute din surse care au fost citate, în textul lucrării şi în bibliografie.

Declar că această lucrare nu conţine porţiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislaţiei române şi a convenţiilor internaţionale privind drepturile de autor. Declar, de asemenea, că această lucrare nu a mai fost prezentată în faţa unei alte comisii de examen de licenţă/diplomă/disertaţie.

În cazul constatării ulterioare a unor declaraţii false, voi suporta sancţiunile administrative, respectiv anularea examenului de diplomă.

Nume și prenume
Răzvan-Gabriel PREPELIUC

Data
09.09.2021

Semnătura
Razvan Prepeliuc

Această pagină a fost lăsată goală în mod intenționat.

Absolvent: **Răzvan-Gabriel PREPELIUC**
Conducător: **Ș.l.dr.ing. Mihai DĂRĂBAN**

SINTEZA PROIECTULUI DE DIPLOMĂ

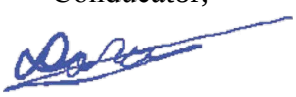
SMART DOG COLLAR

The ideea of a Smart Dog Collar is already found on the market, as special training gears with ultrasounds or with a GPS connected to a mobile phone application. The concept introduced in this project consists of more than one feature that would make the lives of a dog and his owner easier. The GPS module will be a part of this project too, next to an impact detector represented by an accelerometer and an alarm, consisting of two LEDs and a piezo-buzzer, that will be triggered by an impact. More than that, as a separate project, a feature to monitor the food behaviour of a dog will be implemented using an RFID scanner. The tag will be placed on the collar, while the scanner will stay on the bowl, so that an owner can keep track of the food eaten by the dog. The drivers for each component were firstly implemented to test and analyze the functionality of the devices.
Both projects were implemented in C language using Code Composer Studio program, offered by Texas Instruments, as their LaunchPad and BoosterPacks are used. All the parts worked properly and the result displayed on the terminal were as we wished.

Zgardă pentru Câini Inteligentă

Ideea de zgardă inteligentă pentru câini este deja pe piață sub forma unor dispozitive cu ultrasunete folosite pentru antrenament sau cu un GPS conectat la o aplicație mobilă. Conceptul prezentat în această lucrare este alcătuit din mai multe opțiuni care ar putea face viața mai ușoară pentru un câine și stăpânul său. Modulul de GPS va face parte și din această lucrare, alături de un detector de impact, reprezentat de un accelerometru pe 3 axe li o alarmă, formată din două LED-uri și un difuzor piezo, care vor fi activate de un impact. Mai mult de atât, ca un proiect separat, va fi implementată o opțiune de monitorizare a comportamentului nutritiv al unui câine, folosind un modul RFID. Tag-ul va fi amplasat pe zgardă, în timp ce scanner-ul se va afla pe bolul de mâncare, permițând stăpânului să urmărească de câte ori a mâncat câinele său. Driverele pentru fiecare componentă în parte au fost, la început, implementate pentru testare și analiză.
Ambele proiecte au fost implementate în limbajul C, folosind programul Code Composer Studio oferit de cei de la Texas Instruments, ale căror plăci le-am și folosit. Toate componentele lucrează corect și rezultatele afișate pe terminal au fost cele dorite.

Avizul conducătorului
Conducător,



Absolvent,

Razvan Prepeliuc

Această pagină a fost lăsată goală în mod intenționat.

Contents

1 Review in Romanian

1.1 Stadiu actual

Din ce în ce mai multe produse de pe piață ajung să se dezvolte devenind ”inteligente”. Prima dată au fost telefoanele, apoi televizoarele și ceasurile, fiind dezvoltate concepte pentru o mulțime de alte produse (cum ar fi și casa inteligentă). Toate aceste produse se bazează pe integrarea unui procesor sau controler în interiorul obiectului și au ca scop dezvoltarea unui stil de viață mai bun și mai ușor pentru oameni.

Această lucrare are scopul de a prezenta conceptul de Zgardă Inteligentă pentru câini. Acest dispozitiv, în principal, este conceput pentru a-l ajuta pe proprietar să își monitorizeze animalul cât timp acesta este lăsat singur acasă. Însă, se va observa că prin toate opțiunile pe care le conține, observarea câinelui folosind această zgardă va oferi și acestuia un trai de viață mai bun.

Pe piața mondială există în momentul de față câteva astfel de zgărzi, folosite în schimb pentru un singur rol, spre exemplu: zgardă cu GPS sau zgardă pentru antrenament. Deși aceste două dispozitive au foarte multe avantaje, conceptul de Zgardă Inteligentă, ce urmează a fi prezentat în următoarele rânduri, vine ca o inovație completă pe piața produselor pentru animale, având mai multe opțiuni integrate și folosind un microcontroler pentru realizarea tuturor acestor funcții.

Un prim lucru pe care orice proprietar de câine și l-ar dori, în special cei care îl lasă să se plimbe prin curte sau înafara ei, ar fi un detector de impact. În fiecare zi, o mulțime de câini sunt loviți de mașini, cad în puțuri sau canale sau chiar pot fi bătuți de oameni. De multe ori în aceste situații, depinzând, bineînțeles, și de rasa câinelui, și de puterea impactului, animalele nu decedează pe loc, ci sunt lăsate leșinate până când își dau ultima suflare. Bineînțeles, un astfel de eveniment dur nu poate fi evitat, întrucât câinii sunt niște animale curioase, cu dorință de explorare, dar cel mai important, prietenoase și jucăușe. Așadar, oricât de strict ar fi un stăpân, un câine care este singur acasă, va căuta întotdeauna o modalitate de a scăpa de plictiseala statului în casă. Când acesta ajunge în locuri necunoscute până atunci, mirosul și curiozitatea îl vor conduce cât mai departe, lucru ce poate rezulta într-un dezastru, după cum am spus mai sus.

Chiar dacă impactul nu poate fi evitat, viața animalului poate fi. De aceea, zgarda inteligentă va conține un detector de impact, bazat pe un accelerometru pe trei axe și o alarmă, construită pe baza a două led-uri și un difuzor buzzer. În momentul în care accelerometrul detectează un impact, alarma va porni. Acest lucru poate avea două avantaje. Primul ar fi că, în cazul unui impact nu foarte violent, alarma va trezi câinele inconștient, iar acesta se va înlătura singur din zona primejdiei. Al doilea avantaj ar fi că, dacă deobicei, animalele sunt lăsate să moară, deoarece sunt prea rănite pentru a se mișca, această alarmă ar putea atrage atenția oricărui om ce se poate afla în zona respectivă, nu doar stăpânului care îl caută. Ajutorul imediat al unui om avertizat de alarmă ar putea salva o viață.

O altă opțiune ce va fi prezentă la zgarda inteligentă și care se află deja pe piață, este monitorizarea prin GPS. Conceptul de depistare a câinilor a apărut cu mulți ani în urmă, prin microcipare, care în momentul de față este obligatorie. Cu toate acestea, ea nu permite stăpânului o urmărire constantă a patrupedului, ci este benefică doar în momentul în care animalul ajunge la veterinar, iar acesta află toate datele necesare, inclusiv ale stăpânului. Așadar, acesta nu poate fi numit un mijloc de urmărire a câinelui, ci mai degrabă un mijloc de identificare al acestuia. Prin simpla conectare a unui modul de GPS la zgardă, stăpânul va ști întotdeauna poziția animalului său, lucru ce îi poate fi de folos și în cazul unui impact, situație descrisă mai sus.

Ultima opțiune implementată în acest prim stadiu al conceptului se bazează pe disciplina alimentară a unui câine, pentru a-l ajuta să ducă o viață cât mai sănătoasă. Acest lucru necesită monitorizarea cantității de mâncare consumată de animal. Acest lucru se va implementa folosind un scanner RFID și o etichetă RFID. Acest lucru aduce multiple avantaje stăpânului.

În primul rând, acesta va ști în fiecare zi de câte ori pe zi câinele a fost la bolul cu mâncare. O zi în care cantitatea de mâncare diferă rațional față de restul zilelor, poate ajuta la depistarea unei

boli (lipsa hrănii poate însemna afecțiuni ale sistemului digestiv, iar hrana în exces poate fi un prim indiciu în depistarea unei sarcini).

De asemenea, acest sistem de monitorizare a hranei prin scanare, poate fi conectat la un alt sistem inteligent de control al cantității de mâncare. Implementarea unui astfel de bol ”inteligent” poate fi facută prin utilizarea unui controler, care după scanare, verifică după niște date, introduse de stăpân, dacă cățelul poate servi o ală porție de mâncare sau nu. În caz afirmativ, acesta va activa un motor DC ce controlează o ușă, prin care va cădea în bol cantitatea de mâncare exactă pentru o porție, stabilită de proprietar.

1.2 Fundamente teoretice

Implementarea zgărzii va fi facută cu ajutorul plăcii MSP432P401R de la Texas Instruments. Aceasta conține controlerul de bază al proiectului. Inafara led-urilor, toate celelalte componente hardware se află fie pe un BoosterPack, fie se conectează separat la placă.

Led-urile (diode emițătoare de lumină) sunt niște componente electronice, constuite din material semiconductor, care, spre deosebire de diode, care disipă energie sub formă de căldură, produc lumină când trece curent electric prin ele. Placa dispune de trei culori diferite pentru led-uri: roșu, albastru și verde, însă pentru alarmă vor fi folosite doar primele două.

Componenta alături de care led-urile vor forma alarma este un difuzor piezo-electric. Acest dispozitiv este folosit pentru a creea sunete și este alcătuit dintr-un piezo-cristal, amplasat într-o cutie de rezonanță, care își schimbă forma când la bornele lui există tensiune și produce astfel niște unde percepute ca sunete. Acesta este deobicei controlat printr-un semnal PWM.

Fiecare notă muzicală este, de fapt, o undă sonoră cu o frecvență specifică. Astfel, trimițând asupra difuzorului un semnal dreptunghiular cu o frecvență de 49 Hz, acesta va vibra sub forma notei Sol (G). Pentru alarmă, nota creată va fi La#4, cu o frecvență aproximativă de 455 Hz. Acest lucru se realizează folosind un semnal PWM cu factor de umplere de 50% și o perioadă de 2,2 milisecunde.

După ce va fi construit acest semnal prin cod, cu ajutorul unui timer, alarma va fi pusă cap la cap, astfel în cât led-urile și buzzer-ul să meargă timp de o secundă, apoi să fie oprite pentru aceeași perioadă de timp, repetându-se acest ciclu până când procesul va fi oprit de către o persoană prin apăsarea unui buton.

Alarma va fi declanșată de către un impact, detectat prin ajutorul unui accelerometru pe 3 axe. Accelerația este mărimea fizică ce măsoară variația vitezei în timp. Ea a fost definită prima dată de Galileo Galilei, urmând ca Isaac Newton să se folosească de ea în definirea uneia dintre cele 3 mari legi de mișcare, definind greutatea (ca forță) ca fiind produsul dintre masa unui corp și accelerația acestuia. Tot Newton a definit gravitatea ca forța ce atrage corpurile spre centrul Pământului cu o accelerație G, egală cu 9,81 m/s^2 la nivelul mării.

Așa a apărut ideea de măsurare a accelerației cu o nouă unitate de măsură, respectiv G. 1 G reprezintă forța exercitată de masa unui corp când acesta nu este supus mișcării. Cu cât un corp accelerează mai repede, masa acestuia va crește. Spre exemplu, când este supus unei accelerații de 3 G, un corp va simți o forță de 3 ori mai mare decât greutatea sa (corpul va fi mai greu).

Accelerometrul utilizat măsoară accelerații de pană la ±2 G pe 3 axe: x, y, z. Așadar, acesta va citi încontinuu date și le va transmite către microcontroler, iar acesta, când va detecta o creștere bruscă între 2 valori consecutive, urmate de accelerație 0, va declanșa alarma descrisă anterior.

Urmărirea constantă a câinelui este un avantaj foarte mare pentru un stăpân, iar această opțiune va fi inclusă în zgarda inteligentă, prin conectarea la controler a unui modul de GPS. Acesta este un dispozitiv, deja foarte bine cunoscut și folosit în o mulțime de aplicații, care se folosește de sateliți pentru a citi mai multe date. Abrevierea sa vine de la ”Sistem de Poziționare Global” și este folosit pentru a măsura: longitudinea, latitudinea, înălțimea și timpul. Bineînțeles, știind aceste date, și alte mărimi pot fi obținute prin calcul, pornind de la valorile deja obținute (ex: viteza, direcția).

Prima lansare a unui satelit în orbită a avut loc în anul 1957, iar în prezent, după 64 ani, aproximativ 6000 de sateliți orbitează în jurul Pământului, dintre care doar 2600 încă operaționali. Sistemul de poziționare globală (inițial folosit doar de armată și guvern) a devenit disponibil pentru oameni la mijlocul anilor 80, primul telefon cu localizare GPS apărând în 1999.

Pentru o localizare perfectă sunt necesari minim 4 sateliți care să fie deasupra receptorului. În momentul de față este disponibilă o rețea de 28 de sateliți pentru GPS, aceștia fiind amplasați specific în orbită pentru ca la orice moment de timp, din orice punct al Pământului, minim 4 dintre aceștia să fie disponibili.

Dacă sateliții sunt baza sistemului de localizare, cei care trimit datele exacte, modulul GPS folosit este doar un receptor, alcătuit din mai multe părți. Antena preia semnalul care este apoi trecut prin mai multe circuite de filtrare și procesare. Datele sunt trimise către controler care preia și prelucrează datele, oferind utlizatorului latitudinea și longitudinea exactă. Pentru o și mai bună vizualizare a datelor, modulul poate fi conectat la un sistem online de hărți (ex: Google Maps), care va prelua datele și va arăta poziția exactă grafic.

Pentru monitorizarea hranei unui câine va fi folosit un dispozitiv RFID (Identificare a Frecvențelor Radio). Acesta oferă posibilitatea de a scana o etichetă și este asemănător unui scaner de cod de bare, doar că mult mai dezvoltat, întrucât poate fi folosit pentru distanțe mai mari și are o memorie de apeoximativ 100 de ori mai mare.

Un dispozitiv RFID este alcătuit din două componente principale: eticheta și cititorul. Acestea comunică printr-o interfață SPI (Interfață Periferică Serie), care poate fi folosită, după caz, în configurații cu trei sau patru fire. De obicei are patru semnale de bază, iar ceea ce caracterizează această comunicare este posibilitatea de alegere a transferului de date, fie de la maestru la servitor, fie invers, folosindu-se de semnalele MOSI și MISO. Acest protocol poate avea mai mulți servitori, însă un singur maestru.

1.3 Implementare

Pentru implementarea software a zgărzii se vor folosi resursele celor de la Texas Instruments. Baza hardware a proiectului constă în controlerul MSP432P401R, aflat pe o placă cu mai multe componente periferice, iar mediul de programare va fi programul Code Composer Studio (versiunea 10.2.0).

Cele două led-uri folosite pentru alarmă se află chiar pe placa principală și sunt conectate la porturile P1.0 (roșu), respectiv P2.2 (albastru). Pentru o mai ușoară înțelegere și organizare a codului, fiecare componentă implementată în acest proiect va fi configurată într-un fișier sursă separat, fiecare având un fișier de tip header corespondent. Spre exemplu, configurarea led-urilor va fi făcută în fișierul ”led.c”, în care va fi inclus și fișierul ”led.h”.

Prototipurile funcțiilor create vor fi declarate în fișierul header și scrise în fișierul sursă. Pentru led-uri va fi nevoie de trei funcții separate: o funcție de configurare a porturilor respective ca și ieșiri, o funcție de aprindere a led-urilor care va trimite un semnal de 1 logic către cele două porturi și o funcție de stingere care va face fix opusul, trimițând un semnal logic de 0.

Pentru implementarea difuzorului și a accelerometrului se va folosi o placă separată, denumită Education MKII BoosterPack. Aceasta conține o multitudine de componente periferice și senzori, putând fi folosită în mai multe tipuri de aplicații: microfon, LCD color, senzor de temperatură, senzor de lumină, joystick cu giroscop etc.

După conectarea BoosterPack-ului la controler, difuzorul este conectat la portul P2.7. Configurarea sa va fi făcută tot cu ajutorul a trei funcții, cu aceleași scopuri ca și la led-uri, într-un fișier sursă separat. Pentru configurarea unui port al controlerului se vor accesa întotdeauna registrele de direcție, selecție 0 și selecție 1. Acestea vor fi puse în starea 1 sau 0 după cum este indicat în datasheet-ul controlerului pentru configurația necesară (în cazul difuzorului va fi port de ieșire). Celelalte două funcții vor porni, respectiv opri difuzorul.

După implementarea celor două componente, pentru utilizarea lor în sistemul de alarmă, va trebui configurat un semnal de ceas și un timer prin care să activăm componentele. Prima dată va fi setat semnalul de ceas necesar, operație pentru care vor fi utilizate două funcții. Prima va configura portul PJ.0 (un port intern la care nu poate fi conectat alt dispozitiv), iar apoi urmează inițializarea în sine. După ce se accesează regiștrii ceasului folosind cheia (sau parola) de activare, fiecare sistem de ceas ce urmează a fi utilizat va fi setat la o anumită frecvență necesară aplicației.

La această aplicație se va folosi unul dintre cele 4 timere pe 16 biți ale controlerului. Cele trei funcții necesare configurării sale nu diferă ca scop de cele pentru led-uri sau difuzor, însă operațiile lor sunt puțin mai dificile, întrucât sunt mai multe registre ce necesită a fi scrise.

În funcția de inițializare, timerul va fi oprit și se va decide sistemul de ceas ce urmează a fi folosit. În funcția de pornire a timer-ului se va curăța steagul pentru întreruperi și se va hotărî modul și valoare până la care să numere acesta, iar în funcția de oprire doar va fi setat în modul stop.

Pentru a face alarma să funcționeze când este detectat un impact se va folosi o întrerupere de timer, funcție ce trebuie scrisă în codul sursă principal al proiectului. Ca alarma să funcționeze în perioade de câte 2 secunde (o secundă pornită, o secundă oprită) s-a definit un contor ce va număra semnalele de tact, iar în funcție de valoarea sa, starea alarmei va fi determinată.

Accelerometrul este senzorul care va trimite constant date către microcontroler. El are trei pini, câte unul pentru fiecare axă și este conecta la un convertor analog – numeric. Așadar, pinii săi trebuie configurați ca și pini de ADC, lucru ce se va realiza în prima funcție. În procesul de inițializare, acesta va porni, iar apoi registrul de control al memoriei va fi umplut cu rezultate ale conversiei provenite de la cei trei pini, urmând ca întreruperile corespunzătoare celor 3 pini să fie autorizare. Apoi vor mai fi folosite trei funții diferite pentru preluarea rezultatelor conversiilor și plasarea lor în niște variabile ce vor putea fi folosite. Procesul se încheie cu o funcție pentru pornirea conversiei.

Pentru detectarea unui impact se va folosi un algoritm scris în fișierul sursă principal, chiar în bucla de lucru. Acesta presupune calcularea unei valori numită ”delta” inițializată cu 0 și egală cu diferența dintre 2 valori consecutive preluate de la accelerometru pe o axă. Dacă această valoare depășește 1 G, deja se poate pune problema existenței unui impact. Însă dacă acest eveniment este urmat de valori nule ale accelerației pe toate axele, acest fapt indică mai mult decât sigur producerea unui impact și va porni alarma prin apelarea funcției de pornire a timer-ului.

Alarma va putea fi oprită doar prin apăsarea unui buton ce se află, de asemenea, tot pe BoosterPack, fiind conectat la portul P3.5 al controlerului. Configurarea sa nu este deloc dificilă, se folosește doar o funcție pentru a-l seta ca port de intrare și a autoriza întreruperile pe portul său. În fișierul sursă principal se va defini funcția de întrerupere a portului 3, iar în bucla de control se va scrie o condiție ce prevede ca timer-ul să fie oprit când butonul este apăsat.

Pentru urmărirea câinelui se va folosi un modul de GPS extern, denumit NEO-6M. Acesta are disponibili spre conectare 4 pini, însă doar 3 dintre aceștia, respectiv VCC, TX, GND, vor fi conectați la controler, întrucât nu este folosită funcția de receptor a dispozitivului (modulul doar trimite date către controler, nu și primește). GPS-ul operează ca un senzor și procesează niște rezultate. Aceste date primite de la sateliți vor fi primite sub forma unui șir de caractere ce conține toate datele necesare pentru această aplicație și chiar mai mult. Acest șir este denumit ”NMEA”, abreviat de la Asociația Națională de Electronică Marină.

Acest șir trebuie să aibă o structură specifică, cu anumite caractere amplasate pe pozițiile corespunzătoare. El conține mai multe date care trebuie desprinse din șir de către utilizator prin metode software. Aceste informații conțin: longitudinea, latitudinea, timpul, direcția, numărul de sateliți conectați, înălțimea, viteza etc. Însă pentru aplicația curentă, primele două menționate vor fi folosite.

Pentru configurarea și preluarea datelor de la GPS vor fi folosite patru funcții, însă acestea sunt separate de cele pentru afișare. O funcție are rolul de a prelua datele din buffer-ul receptor, altă funcție le transformă matematic în valori zecimale (doar latitudinea și longitudinea), iar o a treia funcție preia aceste date și le pune în niște variabile pentru a fi afișate ulterior. O ultimă funcție are

rolul de a verifica dacă datele primite de GPS sunt corecte, prin compararea șirului NMEA primit cu un standard cunoscut.

Următorul pas este configurarea interfeței de comunicare UART (Receptor-Transmițător asincron universal). Va fi nevoie de două configurații diferite, una pentru comunicarea dintre modulul GPS și controler și alta pentru transmiterea datelor către terminal. Așadar vor fi construite două funții de configurare a porturilor. Una va configura porturile P1.2 și P1.3, pentru comunicarea cu terminalul, iar alta portul P3.2, portul de receptor al controlerului unde este conectat pinul TX al GPS-ului.

Pentru comunicarea dintre controler și terminal se va folosi un baud rate de 115 200. Acest proces este făcut în interiorul unei funcții ce are scopul de a seta mai mulți regiștrii pentru acest baud rate, punând diverse valori obținute prin calcul în regiștrii de control al modulației și al baud rate-ului. De asemenea, mai trebuie ales și sitemul de clock dorit. Pentru comunicarea dintre GPS și controler, în schimb, se folosește un baud rate de 9600. Configurarea și calculele matematice se fac în același mod, diferența dintre cele două funcții fiind că acum este necesară golirea steagului de întrerupere și permiterea uneia din partea receptorului.

Au mai fost create două funcții pentru pornirea și închiderea GPS-ului, funcții care trimit semnal logic 1 (pentru pornire) și 0 (pentru oprire) către registrul de pornire a întreruperilor. O ultimă funcție este folosită pentru trimiterea de caractere către terminal. Aceasta parcurge întregul buffer de transmisie și mută datele din el într-o variabila ce va fi după afișată.

În fișierul sursă principal se construiește separat funcția pentru întrerupere a interfeței UART pe transmisie, iar în bucla de control se află întregul proces de citire și scriere a datelor. Se declară o variabilă întreagă ce conține valoarea flag-ului pentru GPS (va fi egală cu 1 sau 0). Când această variabilă este 1, procesul începe, modulul este pornit, iar datele despre latitudine și longitudine vor fi puse într-un șir (se iau 3 zecimale pentru fiecare valoare) ce urmează a fi afișat ulterior folosind funcția de afișare descrisă mai sus. Între afișări se va folosi o întârziere de 500 de milisecunde (sau de cât dorește utilizatorul) pentru a putea citi datele din terminal.

Partea finală a proiectului conține configurarea modulului pentru RFID. Pentru asta se va utiliza BoosterPack-ul DLP-7970ABP ce are integrat modulul RFID TRF7970A. Acest proiect va fi făcut separat față de ce a fost descris până acum, întrucât pe zgardă se va afla doar eticheta RFID, în timp ce un alt controler cu acest modul se va afla pe bolul cu mâncare. Partea aceasta este cea mai dificilă și complexă de până acum, iar unele procese vor fi repetate.

Se vor configura din nou sistemele de ceas la diferite frecvențe ce vor fi necesare aplicației și de această data vor fi configurate două porturi (PJ.0 și PJ.3), deoarece vor fi folosite și frecvențe mici și mari, deci două cristale separate. Led-ul roșu de pe portul P1.0, folosit și pentru alarmă, va fi configurat și pentru această aplicație pentru a confirma funcționarea corectă a RFID-ului, având scop de debugging.

Următorul pas este configurarea timer-ului. Procesul este identic precum cel pentru alarmă. Timer-ul este inițializat, se setează sistemul de ceas necesar aplicației, iar apoi se crează o funcție pentru pornirea acestuia. Diferit față de configurarea precedentă este faptul că nu mai este nevoie de o funcție separată de oprire, întrucât acest dispozitiv va funcționa în continuu, nici un proces nu îl va face să se oprească.

Cu toate că dispozitivul RFID folosește o interfață SPI, pentru afișarea datelor pe un terminal, asemenea GPS-ului, este nevoie și de o interfață UART. Aceasta va fi configurată identic precum interfața pentru afișarea datelor de pe GPS, folosind porturile P1.2 și P2.3, singura diferență fiind că de această dată va fi folosit un baud rate de 9600. Iar frecvența semnalului de ceas diferă, așadar partea de calcule va fi diferită.

Urmează configurarea interfeței SPI ce conține 8 funcții. Această interfață va fi folosită în configurația de 3 fire, așadar sunt 3 porturi principale ce trebuie setate: P1.4 pentru semnalul de ceas, P1.6 pentru MOSI și P1.7 pentru MISO. Pe lângă acestea, protocolul SPI mai necesită și un port pentru Slave Select (P6.5) și unul pentru pornire (P6.4). După ce s-au făcut toate configurațiile necesare, modulul va fi pornit printr-o funcție ce trimite un semnal de 1 logic către portul pentru enable.

Apoi, se poate trece la configurarea în sine a interfeței SPI. Procesul este unul asemănător cu configurarea UART. Se fac diferite setări accesând registrul Control Word 0 (sistemul de ceas ales, activarea modului sincron, alegerea lungimii între 7 și 8 biți etc.). Urmează apoi două funcții pentru pornirea și oprirea transmisiei de date, lucru ce se face accesând portul de Slave Select. Spre deosebire de alte cazuri, aici transmisia se oprește când portul este setat în 1 și pornește când este setat în 0.

Ultimele trei funcții sunt pentru scriere și citire, mai exact o funcție care scrie datele din buffer-ul de transmisie într-o variabilă, o funcție care citește acele date și returnează buffer-ul de recepție și o funcție care are ca scop tot scrierea datelor, însă într-un alt buffer, fiind folosită pentru șirurile de date.

Ultimul pas înainte de a pune toate funcțiile cap la cap este configurarea RFID-ului în sine. Pentru asta, nu mai puțin de 11 funcții vor fi utilizate. Prima dată va fi nevoie de o funcție care să trimită o anumită comandă, folosită ca parametru, către modul. Acest lucru se face folosind funcția de scriere definită la partea de SPI. Apoi, folosind aceleași resurse, se face o funcție ce are ca scop scrierea unui registru, folosind ca parametri adresa registrului și valoare ce urmează a fi scrisă. Având definite aceste funcții, se poate face pornirea RFID-ului scriind valoarea 0x20 în registrul pentru Chip Status Control.

Citirea se poate face în două moduri, o singură dată sau încontinuu. Pentru aceste cazuri se face câte o funcție diferită. Diferența dintre ele va fi că se modifică valorile hexazecimale pentru comenzi, deoarece al treilea bit din cuvântul de comandă se va schimba în 1, pentru citirea continuă.

O altă funcție importantă se ocupă de detectarea unui câmp extern de frecvențe radio. Aceasta va returna valoarea 1 dacă găsește un câmp extern și 0 în caz contrar. Pentru detectare se folosește registrul RSSI (Indicarea Puterii unui Semnal Receptat). Resetarea RFID-ului se face într-o funcție, scriind în registrele pentru controlul modulației și reglării.

Cea mai complexă funcție are rolul de a stoca datele primite într-un inventar cu trei poziții. Se citesc pe rând registrele pentru întrerupere și FIFO, iar după un timp de întârziere, datele sunt plasate în vector.

Urmează citirea identificatorului unic al etichetei pentru RFID. Pentru a face asta este nevoie de o matrice, care după ce datele au fost stocate în inventar, citește încontinuu datele până parcurge întreg vectorul și le plasează în matricea ce va conține numărul etichetei.

În final, se va inițializa modulul de RFID. Atâta timp cât există un câmp extern de frecvențe radio, modulul este pornit și se scriu registrele de ISO Control, Modulator Control și RX No Response Wait Time.

După aceasta, în fișierul sursă principal, după ce se apelează toate funcțiile de configurare necesare, se lucrează în bucla de control. Cât timp modulul este inițializat, se citește numărul de indetificare al etichetei. Folosind interfața UART, buffer-ul de transmisie preia toate datele din matricea de date și ulterior ele vor fi afișate în terminal.

1.4 Rezultate experimentale

Toate componentele folosite pentru implementarea zgărzii inteligente au fost testate pe rând, separat, iar abia după ce s-a asigurat că toate funcționează corect, a început implementarea generală a codului.

Prima dată a fost testat separat codul pentru aprinderea și stingerea celor două led-uri. După aceea s-a adăugat și codul pentru difuzor, pentru a observa dacă cele două funcționează concomitent. Pentru această testare, accelerometrul nefiind încă configurat, s-a decis funcționarea lor continuă, fără să depindă de vreo condiție. Rezultatele au fost pozitive, alarma funcționând exact așa cum erau așteptările, o secundă fiind pornită, o secundă oprită.

Tot în același regim a fost testat și butonul de oprire a alarmei. După configurarea sa software, adăugând codul lângă cel pentru alarma, s-a observat că apăsarea sa oprește funcționarea acesteia.

Accelerometrul a fost un dispozitiv greu de testat datorită cablului USB scurt folosit pentru conectarea plăcuței la laptop. Rezultatele transmise de accelerometru au fost afișate în terminal folosind un cod separat, pentru a observa funcționarea acestuia. Mișcarea dispozitivului ducea la schimbări ale accelerației pe cele 3 axe, iar cu cât mișcarea era mai rapidă, cu atât diferențele dintre valorile afișate era mai mari.

Pentru testarea modulului de GPS, s-a scris un program care mai târziu a fost copiat în codul sursă principal. Trebuie așteptat circa 30 de minute de când a fost conectat la o sursă pentru a găsi sateliți cu care să comunice. Acest proces e gata în momentul în care un led aflat pe modul începe să pâlpâie. După asta se poate rula codul, iar în terminal se vor afișa coordonatele pentru longitudine și latitudine. Dacă modulul pierde conexiunea cu sateliții, se vor afișa valori nule.

Modulul de RFID a fost testat direct folosind codul final, întrucât acesta era deja scris într-un proiect diferit. După ce codul a fost încărcat pe placa, se observa că led-ul roșu pentru debugging clipește, ceea ce înseamnă că modulul este configurat corect. Apoi pornim terminalul și se observă că se afișează un șir de 16 bytes nul. După trecerea etichetei pe deasupra modului se observă o schimbare în șirul afișat, acesta indicând acum codul unic de identificare al etichetei, exact cum se aștepta.

2 Work planning

Activity	Number of Days	Start	End
Deciding the theme of the project	1	02.11.2020	02.11.2020
Deciding on components to use	7	02.11.2020	09.11.2020
Getting familiar with Code Composer Studio	7	09.11.2020	16.11.2020
Writing the code for the LEDs	14	16.11.2020	23.11.2020
Documenting about the Buzzer	7	23.11.2020	30.11.2020
Writing the code for the Buzzer	7	30.11.2020	07.12.2020
Writing the code for the clock and timer and implement the alarm	14	07.12.2020	21.12.2020
Documenting for the accelerometer	14	21.12.2020	04.01.2021
Writing the code for the accelerometer	21	04.01.2021	25.01.2021
Writing the code for the push-button	3	25.01.2021	28.01.2021
Documenting for the GPS	31	28.01.2021	28.02.2021
Writing the code for the GPS	38	28.02.2021	07.04.2021
Documenting about the RFID	14	07.04.2021	21.04.2021
Writing State of the Art	4	21.04.2021	25.04.2021
Writing Theoretical Fundamentals	7	25.04.2021	02.05.2021
Writing the code for RFID	31	02.05.2021	02.06.2021
Writing the Implementation	25	02.06.2021	27.06.2021
Writing Experimental results, conclusion and review	4	27.06.2021	01.07.2021
Reviewing the code and the documentation	3	01.07.2021	04.07.2021
Preparing the PowerPoint presentation	7	04.07.2021	11.07.2021

3 State of the Art

We live in a world where the word "smart" can be found before any word reffering to an object (e.g. Smart TV, Smart Phone, Smart House, Smart Watch etc.). The concept of "smart" basically reffers to introducing one (or more) microcontrollers and other electronic components in the body of our objects and program them in order to satisfy our needs.

All of the new smart objects developed in the past 10 years had the same purpose, to make an easier and better life for humans. As a dog owner, I often have to deal with some ordinary problems like my dog hiding somewhere in the courtyard, running away from me in the park, eating more than it should and having stomachaches or nausea or even not eating for days because of different disease without me or my family noticing. Dogs are well known to be human's best friend and are always by our side. In order to take care of our four legged friends, I came up with an ideea to make the life of our puppies better for them and for ourselves.

Every day we encounter with the job of letting our dogs alone at home for several hours, because they are the ones engaged to protect the house. This can not be called a problem, but still, many things can happen to our dogs while they are alone, without us knowing. So I started thinking myself of what can I do in order to have at least a bit of control in monitorizing my dog while I am away everyday.

People get easily attached to dogs, so losing him may break your heart. Although dogs are known to be loyal, they are also very playful and curious. Everyday you see on the internet or in the streets ads about missing dogs, a lot of them even offering rewards for any information. In order to try and decrease the number of missing or even killed dogs, I will try and develop the concept of "Smart Collar", which will present more features.

Right now, on the global market, there are some electronic collars, but they are used for training the dogs. They are not used for safety and healthy purposes and most of them have only one feature. For example, there are anti-barking collar`s (**Error! Reference source not found.**) which use an ultrasound based static simulation on more levels. Which means that they detect when a dog is barking and they emit a sound which can be heard only by the dogs to make them stop [1].

Used in trainings, there are collars which use sound, vibration and even electric impulses in order to teach them how to be dutiful and listen to the owner [2]. That being said about these two examples of smart collars on the market, I personally find them pretty harming for dogs. I agree that they can be used in training police dogs for example, but I think a normal dog owner should not use any of these on its own pet.

Figure 3.1 Anti-barking collar

As I said before, the concept I will introduce in my study is more based on creating a better and healthier life for our dogs, especially when we can not take care of them, making an easier life for both us and them.

The concept I have in mind will feature:

- An accelerometer for impact detection
- A luminous-ringing alarm
- An RFID scanner
- A GPS

The need of a GPS used to track our dogs apperead many years ago and, as collars can deteriorate, damage, or in the case of an electronic one, discharge, the transponder chip was invented. Many countries even made it mandatory to microchip every dog. This is a very good method for finding a dog, but it also has a disadvantage. When microchipping a dog, the vet will introduce in a data base the dog's identity (name, birth, vaccines etc.) and most important, the owner's details. So if your dog goes missing and someone finds him and takes him to the vet, he will find out who the owner is and reach out to him. The disadvantage of this is that you, as an owner, have no information of where your dog is until someone else finds him, and at that point it may already be late [3]. In order to make it easier for you to find your dog, attaching a GPS module to its collar may be a better ideea. This feature can already be found on the market, but alone (there is no other "smart" feature on the collar)[4]. Even though the battery may look like a disadvantage, considering that it can last for 15 hours (which is a minimum, considering the battery technology available on the market), it is more than enough for the owner to observe that the dog is missing and to go and find him, without just waiting and hoping for someone else to find it and take it to the vet.

Another important feature included in my concept is the 3-axis accelerometer. This may have more benefits for our dog. First, dog trainers will see this as a good way to measure the acceleration of the dog, see how it improves or test his endurance on different situations (e.g. in the rain, uphill or downhill). But for this, only a one axis is needed. The reason why I choose a 3-axis accelerometer is that I find it very important to find out about powerful impacts or big differences in the acceleration.

For example, if the dog gets hit by a car or by a man, or even fights with another dog, the accelerometer will get a high value on an axis, and then get zero if the dog can no longer move. This can be a life and death situation and I am sure that most of dog owners will want to know if something happened to their dog. Also, in order to make use of the third axis, we can detect if our dog fell off somewhere. In this case, we will first get a very high value on an axis, but show zero after that.

But only finding out about your dog's situation may not be enough to save him, I implemented an alarm for the collar. This alarm includes a buzzer and two LEDs. The buzzer will beep and the LEDs will blink when the alarm is activated, meaning that in one of the cases described above. This has multiple advantages, not only that it may help the dog wake up if it blacked out, but it helps you (or other people) find him if he is stuck somewhere. This feature of alarm and accelerometer may not look very interesting at first, but it could save your dog's life by monitorizing its movement.

The last feature I intend to put in my work, is the RFID scanner. This can be used in order to count the times our pet goes to the food bowl to eat. People have a strict, but healthy plan for everyday food. They take three meals a day, most of the time at the same hours, taking care of the weight of the food, not going over the limit in order to have a healthy lifestyle. So why not try to do the exact same thing for human's best friend?

The concept is pretty simple. An RFID (Radio-Frequency Identification) is a method used for identification without contact, through radio waves, using tags. A more complex example of an RFID application is the barcode. By having an RFID scanner on the collar and a tag on the food bowl, not only do we intend to count the times our pet goes to eat, but with a more complex concept, made of an automated slide system, connected to the collar, the owner can decide the amount of food the dog is allowed to eat while it is alone at home. Because most of the people leave the bowl full of beans when leaving home in the morning, the dog can eat it whole and then sleep all day, which although is a pleasant lifestyle, it is not a healthy one at all.

Also, same as people, dogs may have eating disorders caused by healthy problems, such as stomachache, nausea or even depression. As a person who is gone at work the whole day, you may not notice that your dog has not eaten at all the whole day, or even more. So by having this feature, counting the times he has gone to the bowl may be an extra-advantage, helping you find out if your is healthy and eats regularly.

After analyzing the benefits of the “Smart Dog Collar” and the similar products on the market, I realized that in order to make it possible and useful, it must have a low power consumption. As mentioned in the previous paragraphs, I want to make a device that can make a dog’s life better. In order to do that, the collar should not be heavy, nor very big and confortable. And as we want to implement more features to it, this include more electronic components, so more space and more power consumption. So we will need to make a compromise and analyse all the components to decide on how we want to make the collar both useful and lasting.

4 Theoretical Fundamentals

4.1 Luminous alarm

In order to implement the ideea oa a luminous alarm to lead us to our dog, I will use two LEDs of different colours and a piezo-electric buzzer.

An LED (Light Emitting Diode, Figure 4.1) is a device made from a semiconductor material which creates light when current flows throught it. They were first invented in September 1961 by James R. Biard and Gary Pittman from Texas Instruments. The process behind their functionality is called electroluminescence. The energy resulted is represented by photons, a type of elementary particle that appear from recombination of electrons and holes in the semiconductor material. Light emitting diodes, as the name says, are basically diodes, having a P-N junction to allow current flow only in one direction. The main difference is that the energy dissipated is transformed in light, instead of heat.

Figure 4.1 LEDs

As said above, I am going to use two different color LEDs: a red one and a blue one. There are two main properties that give an LED its color :

- Concentration of dopant
- Wavelength (and frequency) of light

The first LED's ever made were the infrared ones (IR), still used today in remote-control application (e.g. TV remote). Their main advantage is that this type of light can not be seen by the human eye, because it has a wavelenght of 760 – 1000nm and the visible spectrum of the human eye is in the range 380 – 750nm.

The colours that I am going to use are red and blue. Red is the colour closest to the up margin of the visuable spectrum, having wavelengths from 630 to 750nm and is also the colour with the largest wavelength interval (120nm) from the spectrum of the visible colours. Blue has wavelengths up from 450 to 490nm (40nm interval) and is in the opposite part of the spectrum, closer to ultraviolet light than infrared [5].

The visible spectrum of the colours contains the rainbow colours with wavelength intervals increasing from violet to red, ultraviolet and infrared being types of light that can not be seen.

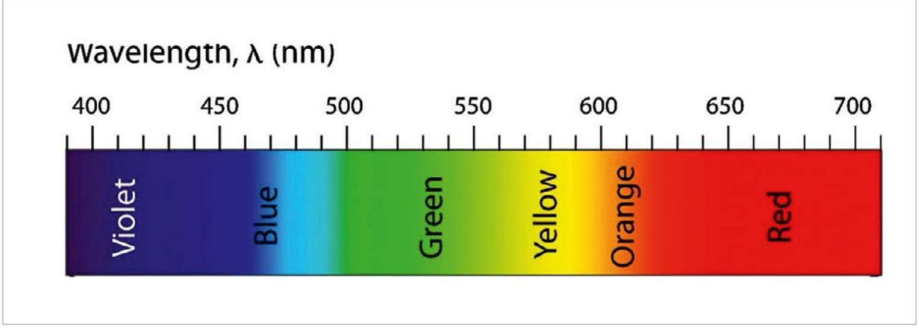


Figure 4.2 The visible spectrum of colours

The LEDs for our Smart Collar will be controlled through digital impulses. When the LED gets a high impulse (1), current will flow through it and it will produce light, while when getting a low impulse (0), it will do nothing. LEDs are mostly controlled through PWM signal (Pulse Width Modulation), as they can switch on and off at very hitgh frequencies that the human eye can not even observe the switch. But in my application, I only need to make them blink simultaneous with the alarm beeps, which will be a period of one second. This can be done either by sending a PWM signal to the LED with a frequency of 1Hz, or by sending "1" (for turn on) and "0" (for turn off) and use a timer to create a delay of one second between the switches.

The next component in our alarm is the piezo-buzzer (Figure 4.3). This is a device used to generate sounds (or tones). It is made out of a piezo crystal put inside a resonant box. The crystal changes shape when voltage is applied to it, creating a pressure wave which is percepted as sound.

Figure 4.3 Piezo-electric buzzer

The generated sound (tone) can be modified by changing the frequency of the signal sent to the buzzer, or the voltage (a higher voltage means a bigger deformation of the crystal)[6].

They can be easily controlled through a PWM signal, meaning that everytime the signal will be on the "1" level, the buzzer will beep. By sending a signal with a high frequency, the buzzer wil beep many times during a short period, giving the human ear the impression of a constant sound.

As for light, we have seen that every colour has a corresponding wavelength in the visible spectrum, for sound, every note (tone) has a different frequency. The audibly interval of frequencies is from 20Hz to 20KHz. Musical notes start from C0, which has a frequency of only 16.35 Hz and is the lowest note ever percepted. As notes are higher, their frequency is also higher, up to even the 8th octave (B9 has a frequency of 7.9KHz)[7].

	Octaves				
Notes (Hertz)	**1**	**2**	**3**	**4**	**5**
C	32	65	130	261	523
C#	34	69	138	277	554
D	36	73	146	293	587
D#	38	77	155	311	622
E	41	82	164	329	659
F	43	87	174	349	698
F#	46	92	185	369	739
G	49	98	196	392	784
G#	52	104	208	415	830
A	55	110	220	440	880
A#	58	116	233	466	932
B	61	123	246	493	987

Figure 4.4 Frequencies of most of the musical notes

In my application, I want my buzzer to create a frequency of 455Hz (approximately A4#). By using the well known formula for frequency calculation (Equation (1)), we determine the time period of the beeps of the buzzer, being equal to 2,2 miliseconds.

$$f = \frac{1}{T} => T = \frac{1}{f} \quad (1)$$

As said above, the buzzer works by alternating the voltage sent to it between low and high. Same as using a PWM signal with 50% duty cycle, the buzzer will beep for 1,1 miliseconds and the stop for the same time, repeating this process. In this way, the note A4# is created. Then, in order to make the sound last for 1 second, we need to calculate how many signal periods are in 1 second (Equation (2)). The result indicates that after 909 periods, we need to stop the buzzer.

$$\frac{1000ms}{1{,}1ms} = 909{,}09 \quad (2)$$

Basically, what this whole process does is that, when activated, the buzzer will beep for 1 second, then stop for the same time, repeating until we stop it. Also, when the buzzer will beep, the two LEDs are also going to turn on. As for some people who may think that this alarm can be very annoying for the dog, its purpose is to activate only when data from the accelerometer indicate that something bad happened to our dog (e.g. he got hit very hard and did not move after that).

4.2 3-Axis Accelerometer

Acceleration represents the variation of the velocity with respect to time, meaning that it can change with speed or direction. For example, an object moving in a straight line with a constant speed will have 0 acceleration, while another object moving very slow, but in a circle, will have a positive acceleration [8].

Acceleration's mathematical formula is written in Equation (3)**Error! Reference source not found.**.

$$a = \frac{\Delta v}{\Delta t} \quad (3)$$

The first physician to state the concept of acceleration was Galileo Galilei (1564 - 1642) who found out that the distance traveled by an object is proportional to the squared time of the movement. After him, Sir Isaac Newton (1643 – 1727) studied the concept of acceleration and stated three famous laws of motion, the second of them including the term of acceleration. The second law said that acceleration appears due to an unbalanced force acting upon an object. In other words, after proving that gravity is a type of acceleration, Newton wrote the mathematical formula of the law (Figure 4.5), saying that weight (reffered to as a force) is created by the acceleration of a mass by gravity [9].

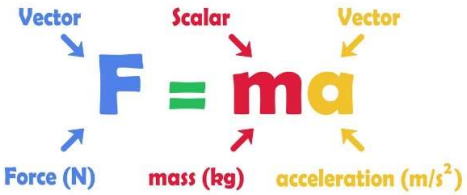


Figure 4.5 The mathematical equation of Newton's 2nd Law

As Newton stated, gravity is the acceleration caused by the force that attracts objects to the centre of the Earth and is equal to 9,81 m/s^2 at the level of the sea. By knowing this, more types of acceleration were defined, but all of them strictly respecting the definition of acceleration. For example, the linear acceleration refers to the case when an object's velocity is changing over time, but the direction of the movement is the same (linear reffering to "in a straight line"). Also, the circular acceleration is the opposite, reffering to the acceleration when an object changes its movement direction, but keeps its speed constant [9].

An important term when measuring acceleration is the "G" force. One G represents the force exerted by an object's own weight, when no movement is involved. Acceleration can influence this force, which will result in an increase in the body weight. A fast acceleration will lead to a greater force and the object will no longer be subdued to only 1 G. For example, if an object that is stationary, starts moving with a very fast acceleration, it can experience a force of 3 G's, meaning that its body will feel a force equal to three times its weight.

Being exposed to a very high acceleration can damage the body of an animal, as it can make the blood go away from the brain and even cause death if it is exposed for a longer time. On a roller coaster, for example, accelerations of 5 to 6 G's can appear, but they only last for a few seconds in order to not harm the humans. For fighter pilots, special suits were designed to keep the blood pressure in the upper part of the body, in order to prevent harmness when exposed to accelerations of 8 to 9 G's [10].

Dogs accelerate faster than humans when it comes to running. There is no way of thinking that a dog will run and accelerate so fast that its blood pressure. Then why would we want to monitorize the acceleration of our dog?

First of all, big accelerations happen on strong impacts. Meaning that if a dog gets hit by a car, a bike, or even a man, a variation in acceleration will appear. By tracking the movement of our dog, we will be able to observe the differences in its acceleration. After an impact, he will not move and this will trigger the alarm previously described.

Secondly, we make use of the z-axis in case the dog falls somewhere. A big acceleration downside followed by no movement from our dog will again trigger the alarm, in order to try and wake him up or get attention to people until the owner arrives to offer help to the dog. This can be a life and death situation and it is always better to be safe than sorry.

Finally, the accelerometer on the collar could be used in dog training. Measuring everyday how fast your dog is can offer you information if you dog evolves or not, helping you to know if you training have results or no. Competition dogs must be very well trained and monitorizing its movement might be a great start for a great training. Also, body motion and posture helps you understand a lot about an animal's behaviour [11].

Acceleration will be measured using a silicon linear accelerometer. This sensor is a Micro Electro Mechanical System (MEMS), meaning that it is made using microfabrication techniques and is pretty similar to an integrated circuit (IC). The basic sensor in an accelerometer can be a silicon capacitor or a piezoelectric crystal, which are integrated in a micro-circuit. The working principle behind this device is that when acceleration appears, a displacement of the silicon structure appears, leading to a change in capacitance. If that happens, voltage, which is proportional

to acceleration, will also change, indicating the acceleration on one axis [12].

Figure 4.7 Accelerometer Adxl335

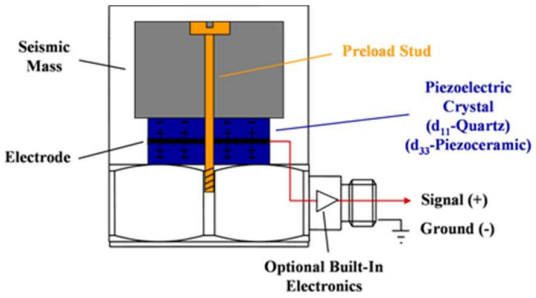


Figure 4.6 Internal Structure of an accelerometer

4.3 Global Positioning System (GPS)

If tracking a dog's movement lets you know a lot about his behaviour, tracking down his actual position is a very useful feature. Animals, in general, can easily run away from home. From my experience as a dog owner, I know that dogs are attracted by smell. A new pleasant smell will make them so curious, that they may try to find any posibility to leave the yard, not to mention the male dogs that will do anything in their power to go to the female dog in the neighbourhood.

Our four-legged friends are genetically made to run and explore. Their well developed smell combined with their curiosity can be a great influence in them running away from home. Adding the fact that the same curiosity and exploring instinct can also be found at cats, the chances that a cat shows up in your courtyard are pretty high. Your dog's instincts will activate and everything he will wish for at that time will be to catch the cat.

Also, boredom is a condition that can appear to animals too. When left alone at home, dogs might get bored of playing on their own with the same toy for hours. That is a reason they might go in search of new territory. The latter, can, of course, be prevented by providing "the three E's": exercise, enrichment and entertainment. By offering them a good and long walk once a day, playing with them 20 minutes a day and petting them when you get the chance, you will avoid your dog getting bored. But this may not be the case of every dog owner [13].

As not everybody has the necessary time to spend with their quadrupeds friends everyday and the other reasons previously mentioned for dogs running away from home can not be so easily avoided, knowing your dog's exact location might free you from a lot of stress. Using a gps module on the Smart Collar will let the owner always know where his dog is. So if you ever get home and realise that your dog is missing, all you have to do is look on the map, get its location and go get your friend back home.

A GPS is an electronic device used measure four units:

- Longitude
- Latitude
- Height
- Time

By monitorizing an object or a person using a GPS, one can also get other information (e.g. direction, speed etc.), but these values are obtained from the other four main units [14].

In order to measure these values everywhere around the globe, GPS modules make use of satellites orbiting the Earth. In order to get the position of an object, a minimum of four satellites is required.

The first satellite ever sent in space was called Sputnik (Figure 4.8 Sputnik, the first satelliteFigure 4.8), and was launched in 1957 by the russians. After realising that the frequency of radio signals sent by the satellite were changing with its movement, the first thoughts of getting the

exact location of a receiver from Earth appeared. So only one year later, in 1958, the first global satellite navigation system, called Transit, was made. After only two more years, in 1960, the first satellite for Transit was launched and could correctly offer navigation information to the military. By the end of 1968, a group of 36 satellites were orbiting the Earth. After 28 years of operating, in 1996 Transit was taken over by the GPS.

In 1978, the first Block-I GPS satellite was sent in space and after 5 years, the U.S.A. announced that when the system will be complete, all the people could make use of the GPS. Nowadays, there are around 6000 satellites orbiting our planet, out of which only 2600 are still operational. Out of those, 1000 satellites are used for communications and only 97 for navigation. The oldest satellite to still be operational was launched in 1991 [15] [16].

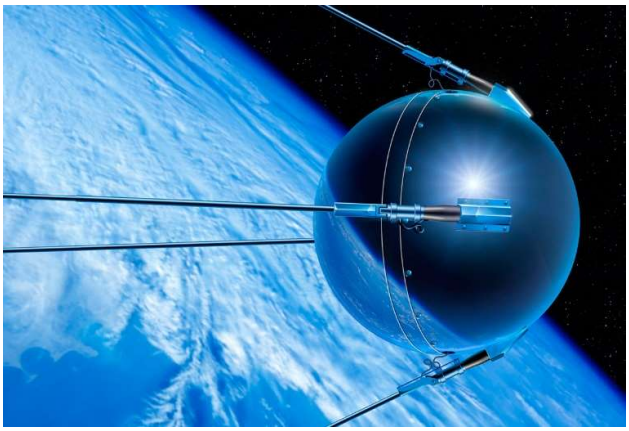


Figure 4.8 Sputnik, the first satellite

In these days, every phone, watch, car and even bikes include GPS systems not only for navigation, but also for tracking. The first mobile phone to ever have GPS appeared in 1999 and was manufactured by Benefon [17]. Airplanes started using GPS in 1994 and the first GPS system for automotive was introduced by the japanesse brand Mazda in 1990, on their car Eunos Cosmo [18].

Satellites use a 1.5 GHz frequency to send its position to Earth. A signal transmitted by a satellite takes aproximately 70 miliseconds to travel perpendicular to the surface of the Earth. In order to get a position on the ground, all you need is an accurate clock in order to compare the travel time of the signal sent by the satellite with the on board clock time. After that, the distance can be calculated by multiplying the travel time with the speed of light. This is how position in a one dimensional space is calculated. In order to get a position in a three dimensional space, four satellites are required [14].

Equation (4) represents the formula to compute the distance to the satellite, where:

$$S = \tau \cdot c \tag{4}$$

- S is the distance to the satellite
- τ is the travel time of the signal
- c is the speed of light, equal to 300 000 km/s

In order to obtain a position on a two dimensional space, two satellites are required. The distance to both of them is calculated using **Error! Reference source not found.**. Then two circles are drawn, the satellites being the center and the calculated distance being the radius. The position of the receiver will be at the bottom intersection of the two circles.

But if we need only one satellite to determine the position in a one dimensional space and two satellites for a two dimensional space, then why would we need four for a three dimensional space?

In reality, any information regarding a position on Earth must be three dimensional, meaning that also the height must be calculated. In order to get the exact position of a receiver, the transit time (travel time) must be very precisely calculated (a delay of 1μs will result in an error of 300m). In order to avoid errors and have correct measurements, a fourth satellite is needed (for a fourth variable, the time error). The error would also appear when trying to determine a position on a two dimensional space, but even using only two satellites for that, the error of the position would be within 5 to 10 meters [14].

At this moment, there are 28 GPS satellites orbiting the Earth, placed in such a way that at any time, there would be minimum four satellites above the position of a receiver. In other words, they are strategically placed so that GPS could be available in any place on Earth. New satellites can be launched anytime, and old one could be dismissed, as the usage time of GPS satellites is about 7.5 years. They use solar panels for charging and always change orientation in order to face the sun. They are about 5 meters long, but their solar panels have an area of 7.5 m^2 and weigh aproximately 2 tonnes (1 tonne for the older satellites) [19].

Figure 4.9 A GPS satellite

The entire Global Positioning System is composed of three segments:

- The Space Segment – all the 28 functional GPS satellites
- The Control Segment – ground station used in monitoring the Space Segment
- The User Segment – user hardware and software

The Space Segment is at a height of 20180 km above the Earth and satellites take about 12 hours to complete their orbit, but adding the rotation of our planet, a satellite will be in the same position after a day (more precisely, 23 hours and 56 minutes). Satellites send signals to the user using two channels (L1 and L2). These signals contain information encoded using phase modulation and three types of code [19].

The first type of code is course acquisition (C/A) code, a pseudo-random sequence repeating with a frequency of 1KHz. This code has information about the time when signal was transmitted and is sent using the L1 channel.

Next is the precise (P) code which contains the same type of information as the course acquisition code, but is used for more precise positioning. If the previous code repeated every 1ms, this code has a repeat cycle of 267 days. It can be found on both channels.

The last code is the Navigation Message which needs 30 second to be transmitted and contains information about orbital parameters, satellite health status, ionosphere and clock corrections. It is sent using the L1 channel [19].

The Control Segment is made up of five monitor stations placed near the equator, having the duty to monitor the signals sent via the two channels and a master control station in Colorado, U.S.A. Apart from monitoring the satellites, the Control Segment can also control and manouvre them into another orbit [19][14].

The User Segment consists of the software developed for positioning and navigation and also the hardware devices used (GPS modules).

Data sent by the GPS satellites and the time of the transmission are very precise thanks to the atomic clocks used. These are the most accurate type of clocks ever made, working at atomic level and being able to cool the atoms to nearly absolute zero (−273,15 °C). But still, due to the receiver clock, errors in the distance might appear. This error is called Pseudo Distance (or Pseudo Range). In order to have a precise measurement of the position of a receiver, we must first calculate the error.

$$\Delta t_{measured} = \Delta t + \Delta t_0 \tag{5}$$

First, we measure the time by adding the time difference between the satellite clock and receiver clock. This is represented in equation (5) [14], where:

- Δt is the travel time of the signal from the satellite to the receiver
- Δt_0 is the difference between the two clocks

$$PSR = \Delta t_{measured} \cdot c = (\Delta t + \Delta t_0) \cdot c = R + \Delta t_0 \cdot c \tag{6}$$

Next, the error in the distance can be calculated using equation (6) [14], where:

- PSR is the pseudo-range
- c is the speed of light
- R is the distance between the satellite and the user

$$R = \sqrt{(X_{sat} - X_{user})^2 + (Y_{sat} - Y_{user})^2 + (Z_{sat} - Z_{user})^2} \tag{7}$$

The distance between the user and the satellite is calculated in a Cartesian System like in equation (7) [14], where:

- X_{sat} , Y_{sat} , Z_{sat} are the cartesian coordonates of the satellite
- X_{user} , Y_{user} , Z_{user} are the cartesian coordonates of the receiver

So far we have discussed about the transmitter of the GPS signal, the satellites, but the more important part of the system is the the device which we will include in the smart collar, the receiver. A GPS signal receiver consists of more parts (or stages).

First of all, an antenna is needed to receive the signal. Because the signals coming from the satellites are very weak, a low noise amplifier (LNA 1) is placed right at the output of the antenna. The signal is then filtered to reduce the affects of signal interference and then enters the HF stage, which includes more circuits (Amplitude Gain Control, 2-bit ADC, Local Oscillator etc.). The next stage consists of the Signal Processor, this and the HF stage being the special circuits in a receiver. The processor has its own clock (RTC). Data is then sent to a controller which gives us the exact results of position and time and sends the result to a display. Of course, in order for the GPS receiver to work, a power supply is needed [14].

The GPS communicates with the microcontroller through an UART protocol. This is a type of serial communication and has three main components: the transmitter, the receiver and the baud rate.

To conclude the feature of tracking, the Smart Dog Collar will include a GPS receiver that will monitor a dog all the time when the owner is not at home. The GPS module will give us the exact latitude and longitude every second, but it can also be connected to an online system of maps, to be easier to track it down. In this way, a dog owner will phisically see the position of its pet, but for the moment, I will only implement the tracking feature by displaying the latitude and longitude

(by simply copying the coordinates on Google Maps, the exact position of the dog will be displayed on the map).

4.4 Radio Frequency Identification (RFID)

Tracking your dog is a good way of knowing that he is fine all the time. But the concept of Smart Collar is trying to offer him a better lifestyle, close to a human's. So only monitoring it in order to be sure that he is safe, may not be enough. In order to improve the utilization of the collar and give our dog the best life conditions, I will implement an RFID tag, to control the amount of food served to the dog every day.

The concept is pretty simple. A tag will be attached to the bottom side of the collar, right on the neck of the dog and a scanner will be placed on the bowl of food. In this way, we will keep track of the number of times our dog went to the bowl to eat. Most of the humans respect a food regime of meals a day and anthletes even measure the exact amount of calories in their food, to maintain a good lifestyle. The number of calories a dog may need to have enough energy and maintain a good weight varies, of course, after the race, but it can be calculated using equation (8), where N is the number of calories per day and m is the dog's body weight[20]. For example, a dog with a mass of 10kg will need aproximately 400 calories a day.

$$N = 70 \cdot m^{\frac{3}{4}} \tag{8}$$

After knowing exactly the amount of food your dog needs, you will be able to control it using the RFID. Every time your dog approaches the food bowl, the tag will be scanned and if the bowl is empty, an electric controlled door will allow an exact quantity of food to fall in the bowl. Also, the system may be connected to a scale in the bowl, to be sure that the dog has eaten all, before giving him more. For example, you divide the amount of calories a day in three meals. When the dog gets hungry, he will go to the bowl, the tag will get scanned and he will enjoy his meal, but no more than quantity decided by you. A counter will be implemented in order to keep count of the times he asked for food and you will know exactly how hungry he is. Monitoring its eating behaviour not only helps you to keep him healthy and not be oversized, but also helps you know if something is wrong with it. For example, if he suddenly stops eating as much as needed, you will know that he is probably sick and you can immediately take him to the vet.

The RFID technology is developing more and more and it started to replace the old barcode, because it can scan from a bigger distance and has a bigger memory (100 times bigger than barcode memory).

RFID tags(Figure 4.11) can be of three types:

- Passive
- Semi-passive
- Active

Out of these three, the most common used is the passive one because it does not need a power source to operate. It gets power from the electromagnetic waves sent by the reader(Figure 4.10). It consists of only 2 components, an antenna and a semiconductor chip, and some of them could also have a cover to protect them. They are also very practical because of their long life[21].

Semi-passive tags communicate in the same way as the passive ones, the only difference being that they need a power supply in order to keep the memory in the tag.

The active tags also need to be connected to a power supply and are more expensive than the passive ones. Their advantage is that they operate at bigger range.

Also, the memory of the tag may differ. Some have read-only memories that are already programmed and are cheaper, and some have read-write memories in which the programmer can also write and can store more data[22]. The same classification is also valid for the reader devices.

Figure 4.11 An RFID tag

Figure 4.10 RFID reader

The protocol used for RFID communication is Serial Peripheral Interface (SPI). It is a very well known synchronous interface that is mostly used for communication between microcontroller and peripherals (e.g. sensors, ADCs, SRAM etc.). It can work in configurations of both 3 or 4 (more common) wire. It usually has four signals:

- CS – chip select, used to select the slave, when there are more than one
- SCLK – the clock signal of the SPI
- MOSI – master out, slave in (data transferred from master to slave)
- MISO – master in, slave out (data transferred from slave to master)

In order to begin data transmission, the master has to send a 0 logic to chip select (because this signal is active on 0) and select the slave. The advantage of this protocol is that you can simultaneously send and receive data, thanks to the MISO and MOSI signals [23].

Knowing all these theoretical aspects of the RFID, we will be able to implement the eating counter for the smart collar. Although, every dog owner should study by himself the exact needs of food for his pet, and divide it in portions in the best way. Also, because of the difference in the amount of food needed for every race of dog, this prototype of the collar will only include the RFID tag and scanner and display the number of times the dog went to the bowl. The electric door that will allow the fall of the food into the bowl should be programmed different for every type of dog and all the system should also get information from the electric scale included in the bowl (so that the counter to increment only when the bowl is empty).

5 Implementation

5.1 MSP-EXP432P401R LaunchPad

As the main component in this project I will use the MSP-EXP432P401R (Figure 5.1) LaunchPad which is based on the MSP432P401R microcontroller from Texas Instruments. The LaunchPad does not have too many features itself, including only 2 push buttons, 1 reset button and 2 RGB LED's. But I chose to use it for its capability to connect to other booster-pack plug in modules which include a lot more features.

The LaunchPad has a 40-pin header access for connecting to another modules. It includes onboard debug probe for programming, debugging and energy measurements [24].

It also supports EnergyTrace Technology when used with the Code Composer Studio application. This is a new technology implemented by Texas Instruments to measure the current consumption of the MCU. Instead of amplifying the signal of interest in order to measure the current consumption/voltage drop over a shunt resistor, the EnergyTrace Technology uses a software-controlled DC-DC converter to produce the target power supply. The power consumption of the used microcontroller will be the same as the time density of the DC/DC converter charge pulses [25].

The MSP432P401R can be used in many applications for a wide variety of fields such as: automation (e.g. thermostat), metering (e.g. flow meters), measurements (e.g. digital multimeters), health (e.g. smart-watches) etc.

The controller has a package of 100 pins (25 pins on each side), out of which 40 can be used for connecting external components. Also, the same 40 pins will be used in order to later connect the BoosterPacks that I am going to use in this project. All the pins are grouped in 10 ports (with 10 pins each) and will be referred to in this project as "Px.y" where `x` represents the number of the port and `y` represents the number of the pin. All of the pins can have a variety of software roles. So in order to have a software compatibility when connecting a BoosterPack, we will need to carefully analyse the datasheet of both devices and see all the functions for which the device's pins can be configured.

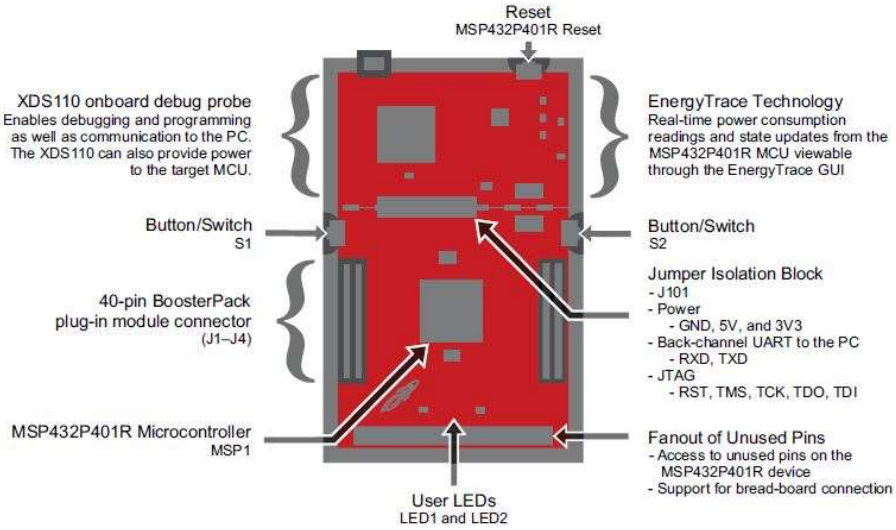


Figure 5.1 MSP-EXP432P401R LaunchPad

5.2 Luminous Alarm

5.2.1 LED configuration

The first thing that we are going to configure in Code Composer Studio are the two blinking LEDs. I chose to make use of the red and blue LEDs already included on the MSP432P401R board, and not use any external hardware. The blue one is on pin P2.2 of the controller, while the red one is on pin P1.0 (LED configurationFigure 5.2).

The first thing to do when starting to write the code is to create a header file (called "led.h" in my code), which will include the prototypes of all the functions that we are going to create for the LED configuration. Next, we need to create a source file (called "led.c") in which we will write the functions needed. In order to do that, we must include the header file previously created in the beginning of the file (using "#include" command). A very important important thing that must be done in every source file created is to also include the "msp.h" and "stdio.h" libraries.

The first thing to do now, after writing the function's prototypes in the header file is to initialise the ports of the two LEDs (P1.0 and P2.2). In order to that, a void function called "led_portINIT()" was created. Both ports must be configured as output ports, and in order to do that, we look in the datasheet of the LaunchPad ("MSP432P401R Datasheet" chapter 6.12, [26]). There says that in order to do that, the port direction register (actioned by writing "DIR") needs to be on logic level "1", while the port selection 0 and 1 registers (actioned by "SEL0", respectively "SEL1") need to be on logic level "0".

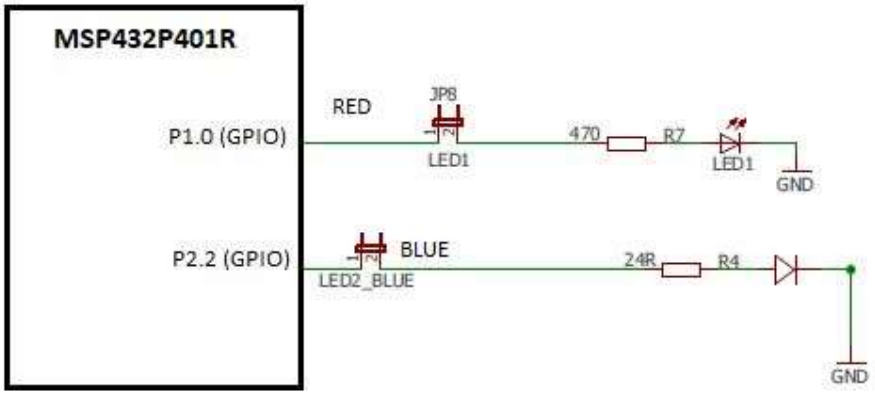


Figure 5.2 LED configuration

In order to write that in C language, we will use the "AND"/"OR" logical functions. First, we write the number of the port we want to configure (P1 and P2 in this case) and access the register that we need by using "->" operator, followed by the name of the register (e.g. SEL0). Then we perform the logic operation that we need with the respective operator ("|=" for logic 1 and "&=~" for logic 0), followed by the number of the pin, which is accessed by writing "BITx", where x is the number of the pin.

After doing this, at the end of the function, the LEDs will be initialized in logic 0, to be sure that they will be turned off when we connect the board to a power source. To do that, the port output register will be accessed (by writing "OUT") and put in logic 0 in the same way as described above.

As our goal is to make an alarm, we will need the two LEDs to blink, meaning that we have to turn them on and off. To do that, two separate void functions, called “led_TurnON()” and “led_TurnOFF” were created.

5.2.2 **BoostXL-EDUMKII Educational BoosterPack**

To implement all the features I have talked about in the beginning of the project, I need to use other boards called Booster Packs. These are just some modules with a lot of features and sensors, but which can not work on their own, because they lack a microcontroller. For this application I chose to take advantage of the many features included in the BoostXL-EDUMKII Educational BoosterPack (Figure 5.3) which is used in a variety of applications:

- Sound applications
- Gaming
- Movement tracking
- Environment description

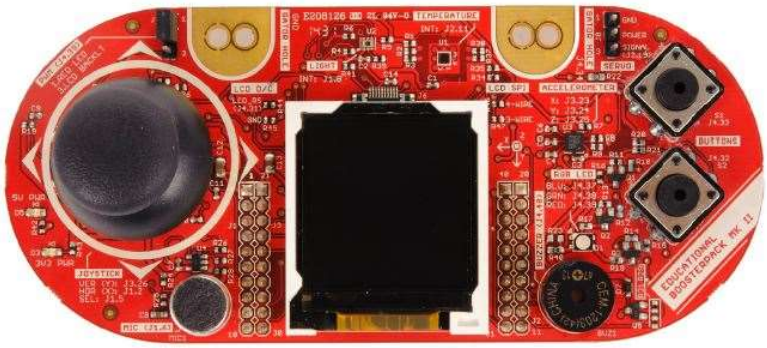


Figure 5.3 BoostXL-EDUMKII Educational BoosterPack

From the many key features of this device, I will use the Piezo buzzer, a device which creates a sound (or a beep) when electric current passes throught it. One of the main applications of a piezo buzzer in our daily lives is the alarm.

The device contains 2 digital sensors which can be used for analyzing the ambient. The TI OPT3001 light sensor and TI TMP006 temperature sensor. The light sensor measures the intensity of light as visible by the human eye. The temperature sensor is an infrared thermopite contactless sensor that measures the temperature of an object without needing a direct contact between them. Both sensors communicate with the controller with an I2C protocol [27].

Another 2 interesting features of the BoosterPack are the Servo Motor Connector and the Accelerometer. The servo motor connector is a 3-pin header to which the user can connect an external servo to be controlled. The accelerometer is a 3-axis analog device (Kionix KXTC9-2050). It is used to measure g-forces. These 2 features can be used in movement tracking appplications.

As mentioned above, the BoosterPack can be used for sound applications. Not only the piezo buzzer can be used for that, but also the CUI CMA-4544PF-W microphone. It has an operating range from 20Hz to 20kHz (almost as the human ear) and is connected to an OPA344 operational amplifier to boost the output.

One of the most exciting features of the device is the ITEAD IM13030001 Joystick which alongside the 2 pushbuttons and the LCD display can be used in gaming applications. The joystick also includes a pushbutton itself. It consists of two potentiometers, each of them being responsible of one axis.

The Crystalfontz CFAF128128B-0145T color 128x128-pixel TFT LCD can represent up to 262K colors and has a contrast ratio of 350. It can be controlled through the SPI interface and can supports up to 20 FPS (frames per second) [27].

As said in chapter 5.1, when initalizing the pins in the code, the user needs to be very careful and look in the datasheet of the devices and analyze the color match between the pins of the LaunchPad and the pins of the BoosterPack. The BoostXL-EDUMKII BoosterPack adheres to the 40-pin LaunchPad as can be observed in Figure 5.4.

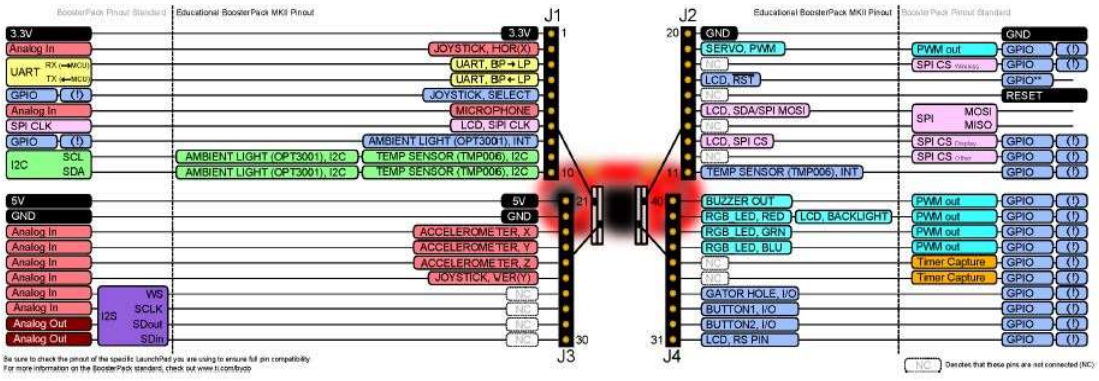


Figure 5.4 Pin mode of the EDUMKII BoosterPack

5.2.3 **Buzzer configuration**

We will treat the code for buzzer configuration in a similar way to the one of the LEDs. After creating the header file ("buzzer.h") and the prototypes of the functions, we start writing the functions in the source file ("buzzer.c").

After connecting the BoosterPack with the MSP432P401R LaunchPad, the buzzer will be on pin P2.7 and it needs to be configured as an output port (Figure 5.5). To do that, a void function called "buzzer_portInit()" was created. After analysing the datasheet, the direction register must be on logic level "1", the Select 1 register must be on logic level "0" and the Select 0 register also on logic level "0".

Same as for the LEDs, as explained in chapter 4.1, the output of the buzzer must be altered between the two possible logic levels. In order to do that, I've created a void function "buzzer_beep()" that will make the buzzer make a sound, by placing the output register on logic "1". In the same manner, a void function "buzzer_stop()" will turn the buzzer off, by putting the output register on logic "0".

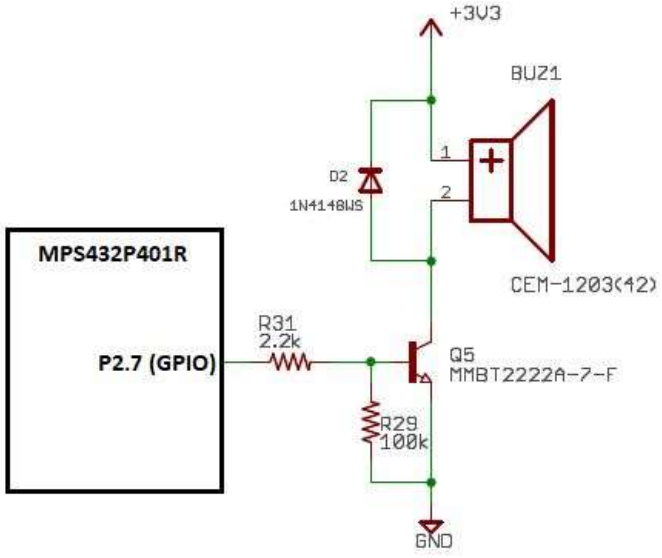


Figure 5.5 Buzzer configuration

To initialise the buzzer and make sure that it is turned off when a power source is connected, instead of just putting the output on "0" logic, inside the "buzzer_portInit()" function, the "buzzer_stop()" function was called. Theoretically, it is the same thing, but we do this in order for the code to be more easily read and understood.

5.2.4 **Clock configuration**

The clock system of the MSP432P401R LaunchPad can operate without the use of other hardware external components and has 7 clock resources [28]:

- Low-frequency oscillator (32768 Hz)
- High-frequency oscillator (1 MHz to 48 MHz)
- Internal digitally controlled oscillator (3MHz and programmable)
- Internal very-low-power low-frequency oscillator (9.4 kHz)
- Internal low-power low-frequency oscillator (32.768 kHz or 128 kHz)
- Internal lowe-power oscillator (25 MHz)
- Internal oscillator (5 MHz)

Also, it has five primary clock systems:

- ACLK – auxiliary clock
- MCLK – master clock
- HSMCLK – subsystem master clock
- SMCLK – low-speed subsystem master clock
- BCLK – low-speed backup domain clock

To configure the clock system, two main functions are needed in the source file "clocksystem.c" (header file named in the same way but with changed extension). First function, called "clocksystem_PortInit()" of type void, has the purpose of choosing the crystal that we want to use, and its functionality (wheter it is needed in crystal mode or bypass mode).

As the best option for this application is to choose the LFXT oscillator, as it supports ultra-lowe-current consumption, the pin PJ.0 must be configured. Port J of the LaunchPad is the internal port where no external components can be connected and most of the pins are connected to the oscillators.

Looking into the datasheet of the LaunchPad([26]), it is confirmed that in order to use LFXT oscillator in crystal mode, the Select 0 register must be on logic "1", while the Select 1 register must be on "0". This is all the operation necessary for port initialisation.

The other function that needs to be written is for the initalization of the clock system ("clocksystem_Init()") and is also of type void. The first thing that needs to be done in this function is to write the key value in the CSKEY register to unlock all the clock system registers. Because the "msp.h" library was included in the header file of the clock system, the defined constant "CS_KEY_VALUE" can be written in the key register (the numeric value of the key is 0x0000695Ah).

As in this application, a programmable oscillator is needed, DCO (digitally controlled oscillator) must be programmed. After choosing the frequency for it (6 MHz in this case), we must write the value 010b in the Control 0 register. Once again, for an easier understanding of the code, the library "msp.h" is used and this value is defined in CS_CTL0_DCORSEL_2 constant, which is written in the register [28]. To mention that in all the future parts of the code, constants already predefined in the "msp.h" library will be used instead of numeric values.

After setting the frequency of the digital oscillator, this must be enabled by writing CS_CTL0_DCOEN in the Control 0 register. To be sure that the register is not overwritten, the "OR" operator is used instead of the normal equal one ("=").

Next, the frequency for the clock systems must be obtained. This next step is done by writing in the Control 1 register. In this application, only SMCLK clock will have a different frequency set. This is done by dividing the frequency of the digital clock to 1, 2, 4, 8, 16, 32, 64 or 128. The

needed frequency is of 3 MHz, meaning that CS_CTL1_DIVS__2 must be written in Control 1 register. To select the source for SMCLK (DCO in this case), the SELS field in the register must be ocuppied by the value 011b, already defined as CS_CTL1_SELS__DCOCLK. And to set the source for the MCLK clock (also DCO), the same binary value is written, but in the SELM field of the register, using CS_CTL1_SELM__DCOCLK [28].

Same as for the Control 0 register, "OR" functions are used to be sure that the register is not overwritten.

The final task for the clock is to create a loop that verifies if every period, the clock signal remains the way it was set and does not get perturbed. The Clock System Clear Interrupt Flag Register is written so that all the fault flags are cleared. After that, the clocksystem key is reseted to 0.

5.2.5 **Timer A configuration**

The MSP432P401R LaunchPad has four 16-bit timers (Timer A) with capture, compare and PWM capability and two 32-bit timers (Timer 32). For this application, a 16-bit timer was used.

After creating the header file for the timer configuration ("timer.h"), in the beginning of the source file ("header.c), two more libraries need to be added ("stdint.h" and "stdbool.h"). Timer A0 needs three functions in order to work properly: one for initialisation, one for start and for stop.

In the first function, called "timer_init()", first of all we must be sure that the timer is stopped at the moment when a power source is connected. To do that, in the control register, the mode control field should be ocuppied by the 00b value or just write MC__STOP. This operation will make sure that the timer is halted. Next, it must be decided what clock is used. This decision must be made after evaluating what clock can handle the needed frequency. As this application does not require a high frequency, ACLK was used. That means that the Timer Source Select Field must be occupied by the 01b value (representing ACLK). This is replaced in the code by TIMER_A_CTL_SSEL__ACLK. The last operation that needs to be done in this function is to enable the timer interrupt. For that, the Capture/Compare Control Register 0 is used. The value is 1b in the Interrupt Enable field or TIMER_A_CCTLN_CCIE command in the code.

Next step is to start the timer. First thing to do is to make sure that the interrupt pending flag is cleared. This operation is done in the Control Register using the "AND" operator in the Interrupt Flag field. The shortcut used in the code is "TIMER_A_CTL_IFG". Next step is to clear the timer writing "TIMER_A_CTL_CLR" in the same register (must be aware to not overwrite it).

The timer has four operating modes: stop mode up mode (counts from 0 to a given value), continuous mode (count from 0 to FFFFh), and up down (count from 0 to a given value and back to 0). As in the initialisation process, the mode control was filled with 00b (TIMER_A_CTL_MC__STOP), now when it needs to be in up mode, the value is 01b and the command is TIMER_A_CTL_MC__UP.

The value to which the timer will count is stored in the Capture/Compare 0 register. As calculated in **Error! Reference source not found.**, there are needed 909 periods of the signal to obtain the required frequency. By taking the value of ACLK frequency and dividing it by 909, the result is aproximately 36, value which must be stored in the CCR[0] register.

Last thing that needs to be done is to stop the timer. This could be easily done by simply putting the timer in stop mode, but this operation may create problems, overwritting some fields from the register. A safer method would be to write the value FFCFh in the control register using the "AND" function .

5.2.6 **Interrupt Request Handler**

The timer needs and interrupt in order to activate. When a condition is met, if the interrupt is enabled, a request will be sent to the controller and the process inside the interrupt will take place.

To do that, in the main source file ("main.c"), the interrupt vector of timer A0 must be shifted with one bit after an AND operation with 31. This operation will be made for all the upcoming interrupt requests. The Interrupt Set Enable Register works only on the first 31 values (1F in hexadecimal) and the shifting is made to enable the interrupt on one specific bit. Next, a function of type void for interrupt request must be written ("TA0_0_IRQHandler") and the first thing that need to be done is to clear the compare interrupt flag. By putting the Capture/Compare interrupt flag field, from the Capture/Compare Control 0 Register on "0" logic, the flag is cleared.

Outside the main function, a volatile integer variable called "counter" was created, that will keep count of the pulses of the timer. More specifically, as it is wanted for the buzzer to beep every 1.1ms (to make the note A#) and keep doing that for 1s and then stop for another second. To keep this count, the variable must be incremented every time the timer interrupt happens.

Next, the processor must verify if the count has reached the value 909 (the number of periods calculated in **Error! Reference source not found.**). Until this condition is fullfilled, the buzzer must beep every 1.1ms. To do that, another condition is made to verify if the counter is even or odd, using the "AND" operator ("&") between the counter value and 0x0001. When the counter value is odd, the buzzer will beep (use the "buzzer_beep()" function), otherwise it will stop (using the "buzzer_stop()" function). Also, on even values, the LEDs will turn.

After reaching the value 909, the counter will still be incremented. The next condition that has to be treated is when the value is between 909 and 1818. As can be observed, if the buzzer and LEDs will be on for 1s and off for another second, it means there is a duty cycle of 50%. So the number of periods to be off is also 909, that results in a total of 1818 periods for a complete cycle. While the counter is between these 2 values, the buzzer and the LEDs will be stopped using the functions "buzzer_stop()" and "led_TurnOFF()".

Finally, if the counter exceeds the value 1818, it will be reseted to 0 and the process wil start again, until another interrupt or command will stop the process.

5.3 Impact detection

5.3.1 Accelerometer configuration

The accelerometer used is implemented on the Educational EDUMKII BoosterPack. In its datasheet, information about the pins connected to it are offered. The pin for the X-axis is J3.23, the pin for the Y-axis is J3.24 and the pin for the Z-axis is J3.25. After, analyzing the pin

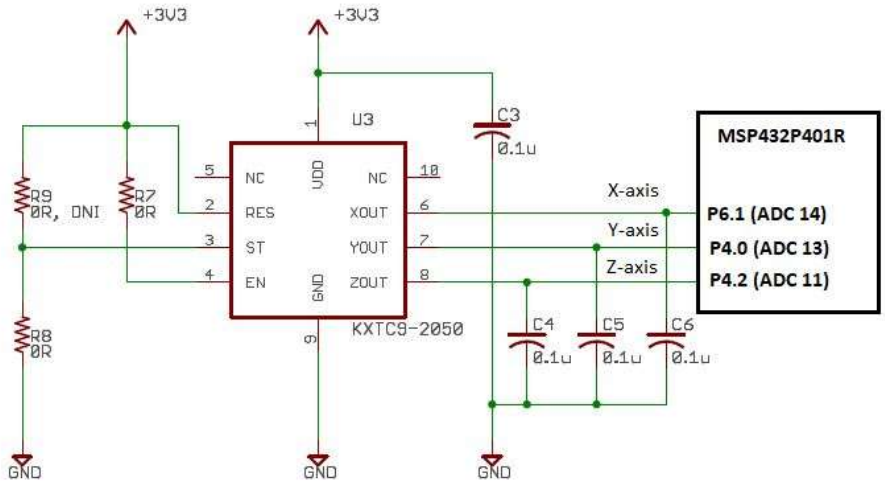


Figure 5.6 Block scheme for the accelerometer

configuration from Figure 5.4 Pin mode of the EDUMKII BoosterPackFigure 5.4, we know what pins from the LaunchPad needs to be configured.

Pin J3.23 from the BoosterPack connects to pin P6.1 from the LaunchPad, pin J3.24 to P4.0 and pin J3.25 to P4.2. All these pins from the LaunchPad are connected to and needs to be configured as ADC pins (as seen in Figure 5.6).

The accelerometer will work under the command of the ADC, the values from every axis being converted and saved in the memory of the ADC. To configure it, six functions will be needed in a new source file, called "accelerometer.c" (the header file has the same named with extension ".h").

First of all, as in the case of the buzzer and the LEDs, the pins must be configured. For that, chapter 6.12 from the LaunchPad's datasheet will be of great use. A void function called "accPin_Init()" is created. The datasheet indicates that for ADC configuration, all three pins must be configured in the same way. The direction register does not care of the logic level, so it can be placed on "1" or "0" or not configured at all. Both the Select 1 and Select 0 register must be on logic level "1".

Next step is the initialisation of the adc, using another void function called "adc_Init()". The sample and hold time for Pulse Sample Mode field in the Control 0 register define the number of cycles in a sampling period. In this case, 16 cycles are needed, so the field should be filled with the value 0010b. This is done in the code by writing the "ADC14_CTL0_SHT0_2" command in the register. The sample and hold pulse mode select is done using the predefined "ADC14_CTL0_SHP" command in the same register. The timer is turned on with the "ADC14_CTL0_ON" command or by placing the value 1b in the on field.

The resolution of the timer is set in the Control 1 register. The posibble values are 8, 10, 12 or 14 bit. The resolution field must be fullfilled with the value 10b for a resolution of 12 bits. In the code, "ADC14_CTL1_RES_2" is written in the register[28].

The Memory Control Register is saving the result of the conversion in the memory. As the accelerometer gives three values, the first three memory places (0, 1, 2) were chosen. To save the result of the coversion from the z-axis in memory 0, the input channel select is used. In the Memory Control Register, the first memory (memory 0) will be loaded with the "ADC14_MCTL_INCH_11" command (11 comes from the pin number of the ADC). Same thing is done for the other two conversions, replacing the number of the memory and the input channel, following to end the sequence using the EOS command in the same register.

Enabling the interrupts for every input pin used of the ADC finishes this function. The register used for that is Interrupt Enable 0 register, where the fields for pins 11, 13 and 14 must be ocuppied with the value 1b. A way to enable an interrupt in the code is by writing "ADC14_IER0_IE11" in the register [28].

Next three functions have the purpose to get the conversion result from the memory and place it in a variable. The functions were called "getAccX()" (or Y or Z for the other axis). Three integer values (valX, valY, valZ) were defined in the header file so they can be used anywhere the header is included. These are the variables that will store the result of the conversion. In each of these functions, the interrupt flag 0 register must have its field corresponding to memory zone used filled with the value 1b to load a result. After this condition is met, the result from the memory zone is put in the corresponding variable.

The last function regarding the accelerometer has the purpose to start the conversion of the ADC. The function is called "adc_StartConversion()" and the only thing that needs to be done here is to activate the Enable Conversion and Start Conversion fields from the Control 0 register.

5.3.2 **Impact Detection Algorithm**

The accelerometer measures g-forces on 3 axis. A powerful impact on a dog would phisically identify as a big difference in acceleration on at least one axis, followed by no movement. This is the case in which the alarm will trigger.

At this point, code in the main source file has to be written. The accelerometer must give values continuously, meaning that after initialising the ADC, the functions that give the g-force values on the 3-axis must be called in a continuous loop. For that, a while loop is used, that will perform the code over and over again until it finds an exist point.

Now, an algorithm to calculate the variation in acceleration must be implemented. For every axis, two variables of type float, called aux and delta, were declared in the main file. Delta is the variable which will store the difference between two consecutive values of the acceleration on one axis and aux will store the previous value. Both variables are first equal to zero to avoid errors in calculation.

Delta is equal with the difference between the value of the acceleration and aux. When connected to a power source, the accelerometer starts and gets the first value. As aux is first initialised with zero, the first value for delta will be equal to the first value given by the accelerometer. After writing delta's formula, the current value of the acceleration is moved in aux. So in the next calculus, delta will be the difference between the new value given by the accelerometer and aux, which stores the previous value. In this way, delta will always store the difference between two consecutive values.

If the value of delta on any of the three axis is bigger than 2 (meaning a difference in acceleration of 2 G) and the following values given by the accelerometer are 0 (OR 1?), it means that something dangerous happened to the dog and the alarm must be triggered. In code, if the two conditions are met, the timer start function will be called. If not, the accelerometer will work normally in continue.

5.4 GPS tracking

5.4.1 NEO-6M GPS Module

The chosen GPS for this application is the NEO-6M module from u-blox (Figure 5.7). It is the simplest GPS on the market, it is cheap and the module comes with a ceramic antenna. It also has an EEPROM memory incorporated and a subtitute battery. The library needed for its configuration can be download from the website where it was bought from.

It has 4 pins: VCC, RX (receiver), TX (transmitter) and GND. Looking in Figure 5.4, the connection between the GPS and the BoosterPack (which is already connected to the LaunchPad)

Figure 5.7 GPS Neo-6M Module

can be easily deduced. The module will be powered from 5V, which means that the VCC pin will be connected to J3.21 from the BoosterPack, while GND will be connected to J3.22 (Figure 5.8 GPS connectionsFigure 5.8).

The transmitter pin from the GPS must be connected to the receiver pin from the BoosterPack (J1.3). There is no need to connect the receiver pin of the GPS module, because there is no data that needs to be sent from the controller to the GPS. In a normal case, the RX pin should be connected to the TX pin of the BoosterPack.

The expectations from this GPS is to get the latitude and longitude and display then on the serial monitor. This step will be done using UART communication. Two different UART protocols need to be configured, one for the GPS to communicate with the controller and one for the controller to display te data on the serial monitor.

5.4.2 **GPS Configuration**

The configuration of the GPS is going to be more difficult than that of the accelerometer or the alarm. First thing that must be done is to define the size of the buffer where the data is stored to 128, in the header file ("gps.h"). Then, a structure that contains all the data given by the GPS, called GPS_t, is created. A total of 24 results can be obtained, two of them being the calculated values that are needed for this application (longitude latitude in decimal).

Data is given after the National Marine Electronics Association (NMEA), a standard data format supported by all GPS manufacturers. The GGA (Global Positioning System Fixed Data) gives results that will be later placed in the next variables:

- nmea_longitude – of type float, stores the longitude
- nmea_latitude – of type float, stores the latitude
- utc_time – of type float, stores the universal time coordinated (at meridian 0)
- ew – of type char, storing the direction for the longitude (east or west)
- ns – of type char, storing the direction for the latitude (north or south)
- lock – of type int, indicates the status of the GPS and can have values from 0 to 2
- satelites – of type int, stores the number of satellites used (from 0 to 12)
- hdop – of type float, indicates the error propagation (Dilution of Precision)
- msl_altitude – of type float, stores the altitude after the mean sea level
- msl_units – of type char, indicates the measurement unit used for the altitude (usually meters)

The GPS can measure even more things, like speed, but the above mentioned variables are more than enough for this application. For tracking the dog, only the latitude and longitude are necessary. An extern structure of this type is then created and named "GPS". Last thing to be done is creating the prototypes for the needed functions.

In the source file, we start by defining three 8-bit integer variables. These are called "rx_data", "rx_buffer" of size 128 (previously defined GPSBUFSIZE) and rx_index.

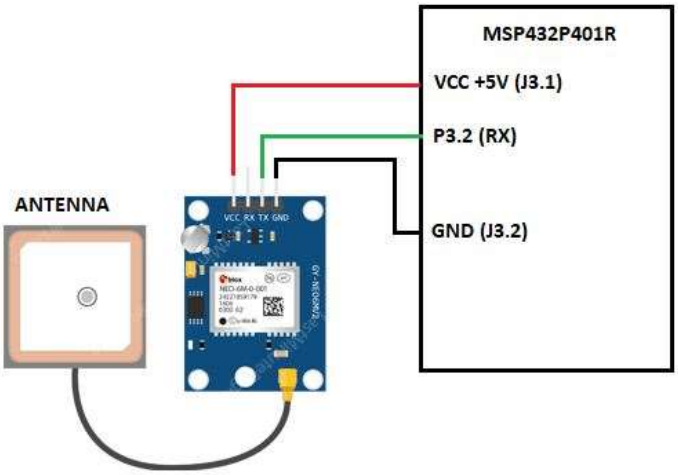
The first function needed is of type void an has the purpose of placing the given data in the receiver buffer and is called "GPS_UART_CallBack()". The content of the Receive Buffer Register is placed in the rx_data variable. Because UART is a serial communication protocol, information is sent bit by bit. Inside an if condition, we verify wheter rx_index is smaller than the total size of the buffer and if the answer is yes, it means that the whole buffer was not yet covered. In this case, the index of the buffer is incremented each time and data is placed in it. Otherwise, if the buffer was fully covered, if the GPS is still sending valid data (verify using GPS_validate function, explained in the next paragraph), the index is put in 0 and the buffer is also reseted to 0 using the "memset" predefined function, which fills the buffer with 0.

Next function, which was already used in the previous one, is called "GPS_validate" of int type and uses a pointer variable to a character called "nmeastr", variable which stores the whole string of data given by the GPS. The purpose of this function is to detect whetever the GPS works properly and verifies if the resulted string is correct. Two string variables of type char ("check" and "checkcalcstr") and two of type int ("i" and "calculated_check") are declared in the beginning.

The first condition that must be met is that the string starts with the "$" character. If the condition is fulfilled, the index (for this function, "i" is the index) is incremented, otherwise the process is stoppes (by writing "return 0").

Other requirements that must be met to assure that the NMEA string is correct are: to have no null values, to be smaller than 75 and to not contain asterisk ("*"). While all this conditions are fulfilled, the checksum is calculated by making XOR operations between the bytes in the string and incrementing the index every time. The checksum is an extra information that is included with each sentence of data. If the index exceeds 75, the program is again stopped. If the asterisk is met in the string, it means that is the end of the data string, and the "check" vector defined in the beginning is filled with the next two bytes of data (and 0 on the last position). In this way, the hex characters are put in the check string. Otherwise, the program is stopped. Data is transformed from decimal to hexadecimal and then a last condition is made. If the values from the 0 and 1 positions in the "checkcalcstr" string are equal with the values on the same position from the "check" string, the function will return "1", otherwise it will return "0".

Next, the data returned in the nmea string for latitude and longitude must be converted to decimal values. For that, a float function, called "GPS_nmea_to_dec" was created. It uses a float variable for the coordinates ("deg_coord") and a char variable for the direction ("nsew"). This function contains basically the mathematical formulas required for this calculus.



First, the degree is calculated by dividing the coordinates degree to 100 and taking the integer part. Next, the decimal part is equal to the difference between the coordinates degree and the calculated degree times 100, divided by 60 (because that many minutes are in a degree). The final decimal result is obtained by adding the the decimal part to the degree. And if the position entered resulted is South or West, the result is retuned negative.

Figure 5.8 GPS connections

Last function needed is called "GPS_Parse" to get the results from the buffer and put them in the respective variables. A pointer to a carachter "GPSstrParse" is used. If the required sentence data is "$GPGGA" (Global Positioning System Fix Data), the data is placed in the coresponding variable from the structure, using references. The GPS can provide the time, latitude, longitude, direction, number of satellites and altitude, but for this application, only the latitude, longitude and direction are needed. The variables "dec_latitude" and "dec_longitude", first declared in the GPS structure, are fulfilled with the final results, by applying the convertion function from NMEA to decimal twice. Once for the latitude and North – South direction and once for longitude and EAST – WEST direction.

5.4.3 UART Configuration

For the UART protocol configuration, a new source file called “uart.c” would be created (header file will have the same name, but with extension .h). For the communication between the controller and the serial monitor, pins P1.2 (as receiver) and P1.3 (as transmitter) will be used. So the first function that needs to be created will be for pin configuration and is called “uart_Terminal_PinInit()”

Looking into the datasheet of the MSP432P401R LaunchPad (chapter 6.12) it can be observed that in order to configure these pins as RX and TX, the Select 0 register should be on logic level “1” and Select 1 on “0” for both pins.

After that, a new function called “uart_Terminal_setup()” has the purpose to configure the protocol by setting the baud rate. First step is to enable a software reset, by accessing the eUSCI Control Word Register 0. Next, the clock source needs to be selected and for this application, SMCLK might be the best solution. The clock source select field of the register must be filled with the value 10b or simply place the command “EUSCI_A_CTLW0_SMCLK”.

Next step is to to do the calculus for the baud rate of 115200. The frequency of the clock signal (3MHz as configured in 5.2.4) is divided with 115 200 multiplied with 16. The result is 1.62 and because it is smaller than 16, the Baud Rate Control Word Register will be filled with the integer part of the result. The Modulation Control Word Register will contain data regarding the modulation stage select and the oversapling mode [28]. The fractional part of the result (0.627) must be looked in Table 24-4 in Technical Reference Manual of the LaunchPad.. In the code, the whole Modulation Control register is filled by shifting the First modulation stage select field with 10 positions. The oversampling mode is enabled and the software reset is disabled, to initialise the serial protocol

The GPS transmits data to the controller also through UART communication, so another interface, A2 will be used (for the first communication, A0 was used). Because the data comes from the satellite to the GPS, only the receiver pin of the controller has to be configured. For that, the Select 0 Register will be on “1” and the Select 1 Regist on “0”. This is included in a void function called “uart_GPS_PinInit()”.

To configure the protocol, a void function, “uart_GPS_setup”, is used. Same as the previous protocol, the Control Word Register 0 is used to select the clock source (also SMCLK) and enable the software reset. The GPS communicates at a baud rate of 9600. So the Baud Rate Control Word Register will store 19. After the software reset is disabled, the receiver interrupt must be enabled. For that, we first clear the receiver interrupt flag from the Interrupt Flag Register. The receive interrupt enable field from the Interrupt Enable Register will store 1 and so the interrupt is enabled.

Because the UART protocol is used only for the GPS application, two more void functions are needed to enable (“Enable_GPS()”) and disable (“Enable_GPS()”) it. All they do is to change the receive interrupt field from the Interrupt Enable Register, by putting 0 to disable and 1 to enable.

Last function that needs to be implemented in the source code of the uart is called “Print_Screen” an uses a pointer to a character as parameter. Its purpose is to sent the character from the transmitter buffer to the data string used as parameter. First of all, the length of the string is placed in an integer variable, called “size”, using the “strlen” function. Also, an index “I” is declared.

After that, a for loop is written to iterate over the string length. And for every iteration, a while condition verifies if there is an interrupt pending or not and place the data from the buffer in the string, until the reach of the end of the string.

The UART interrupt service routine will be defined in the main source file of the project and is called “EUSCIA2_IRQHandler()”. In this function, first of all, it must be verified if there is any receiver interrupt pending. If yes, then the flag must be cleared and all that must be done is to call the “GPS_UART_CallBack()” function (described in 5.4.2), which gets data from the buffer.

5.4.4 Coordinates Display

After setting up the GPS and the communication protocol, all the code must be put together to have a proper functioning of the Smart Collar. First of all, in the beginning of the main source code, some constants are defined in order to help us use a correct delay. This is required so that the controller has enough time to complete its calculations before sending data to the terminal. The Central Processing Unit frequency was defined as SMCLK frequency (3 MHz) and a function "delay_ms(delay)" is calculating the CPU frequency with the given delay and divides it with 1000. In this way, a function using the number of miliseconds as parameter is created. For a smaller delay, in microseconds, the user could define another function and divide the product to 1 000 000 instead of 1000.

After that, an integer type variable ("gps_flag") must be defined and initialised with 0. Its purpose is to store the state of the flag, meaning that it controls the working of the GPS. When the flag is 1, the GPS uart interrupt is enabled and the controller receives data. In different situations, for example after an impact detected with the accelerometer (5.3), the owner may wish that the GPS stop working and display the last coordinates before the impact. This situation is debatable, having advantages and disadvantages. Although in this application, the GPS will transmit data continuously, the flag variable was created to help in future improvements of the application. After that, a string called "altitude_str" will be declared. Its purpose is to store the final data, tha latitude and longitude.

Next step is to initialise the pins in the same way as previously done for the other features. In the main function, "uart_Terminal_PinInit()" and "uart_GPS_pinInit()" functions are called first. After that, for interface initialisation, "uart_Terminal_setup()" and "uart_GPS_setup()" functions are called.

In the same way as for the timer and ADC, the uart interrupt is enabled in the NVIC structure by modifying the Interrupt Set Enable Register.

Now, the whole displaying process happens in the "while(1)" loop, because that makes the controller work continuously when connected to a power source. As said above, the controller receives data when the flag is 1. So if this condition is met, the GPS is enabled. After a delay of 1 second, in which we give the controller time to calculate the result before sending them to the monitor, the gps is disabled. Then the latitude and longitude are placed in the "altitude_str" string with three decimals from the "dec_latitude" and "dec_longitude" variables using the "sprintf" function. To show the result on the terminal, the string is displayed using the "Print_Screen" function. Then the flag can be put in 0 to stop the process. Outside the condition, after another delay (half a second is enough to give the controller time), the flag is again placed in 1.

5.5 Push-Button Driver

As can be observed in Figure 5.3, the EDUMKII BoosterPack also contains two push-buttons, that, along with the joystick and the LCD, can be used in gaming applications. When looking over and debugging the final code for the alarm, we bump over a problem, that once it is triggered by the impact, it can be stopped, only by cutting of the power. If this feature is made in order to help the owner or other people find a blacked out dog, when it is found, it must be stopped.

To avoid this problem, one of the push-buttons from the BoosterPack will be made use of. The second button, which is connected to pin J4.32 was chosen. This means, that pin P3.5 of the controller must be configured as an input port (Figure 5.9).

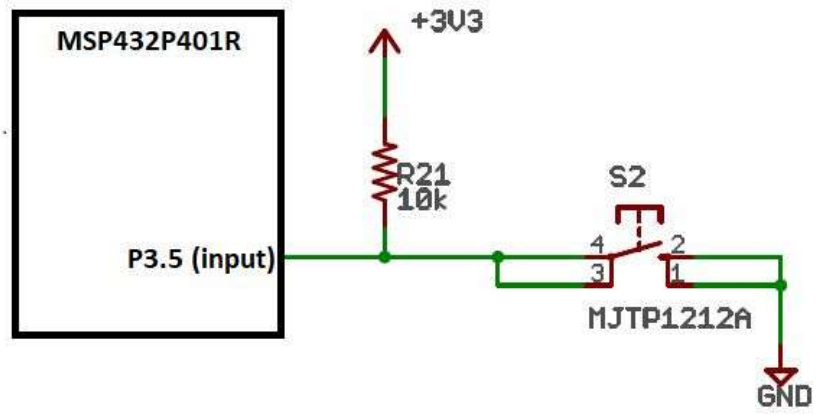


Figure 5.9 Push-button 2 connection

Even though the push-button configuration is pretty simple, another header and source file ("button.h/.c") were created, in order to keep the code as easy to understand as possible. There, only one function is needed for the configuration, which is called "button_Init()".In the datasheet of the LaunchPad it is said that for on input configuration of this pin, all three registers (Direct, Select 0 and Select 1) must be on 0 logic level. In this function also, the port interrupt will be enabled. This is done by placing 1 in the Port Interrupt Enable Register. After that, the interrupt set enable register should be modified to enable the interrupt, but this time, not like in the case of the timer, the ADC or the UART, the Peripheral Exception Number for the Port 3 interrupt request, exceeds 31. It can be still easily enabled by using "NVIC_EnableIRQ" macro extension for port 3.

In the same way as for the UART and timer, the push-button interrupt will be defined in the main source file and is called "PORT3_IRQHandler()". Similar to the UART, it must first be checked in the interrupt flag field if there is an interrupt pending on pin 5 or no. If yes, it means that the button was pushed and an action can occur. After that, the interrupt flag must be reseted to be sure there will be no interrupt pending remained.

Next step is to make the alarm stop when the button is pushed. Because the alarm is controlled through the timer, whenever an impact is detected, is turns on by starting the timer. For the opposite, when it must be turned off, the timer must be stopped. So if the condition of the button being pushed is met, the "timer_Stop()" function is called and will stop the alarm.

5.6 RFID scanner

5.6.1 DLP-7970ABP BoosterPack

The Radio Frequency Identification feature used to count the times a dog went to the food bowl will be separately configurated than the other part of the project, as the Smart Collar will carry the tag, while the scanner and the microcontroller will be placed on the food bowl. For this, another BoosterPack from Texas Instruments, called DLP-7970ABP (Figure 5.10) will be used.

This BoosterPack has only 10 pins for connection with the LaunchPad, meaning that it connects only to the outside pins (J1 and J2). The most important component included is the TRF7970A Transceiver, which can be used for multiprotocol RFID and Near Field Communication (NFC). The desired protocol is chosen by configuring the control registers as needed. An SPI (Serial Peripheral Interface) is used to allow the booster-pack to communicate with the microcontroller unit. The device supports voltage from 2.7V to 5.5V [29].

Figure 5.10 DLP-7970ABP

5.6.2 Clock System for RFID Configuration

The clock system will be configured in a similar way to the clock used for the alarm. First, the clock ports must be initialised. This time, port J.0 and J.3 will both be used, because both high and low frequency pins are needed. Register Select 0 must be on logic level 1 and Register Select 1 on 0, for both ports. This configuration is made in a void function "clocksystem_PortInit()" in the souce code "clocksystem.c" (the names of the files do not need to be different from the previous ones, as they are created in a different project).

Next step is to configure all the clock modules in the void function "clocksystem_Init()". The process is similar to the one described in chapter 5.2.4. First the key is used to obtain access to the registers. This time, the digitally controlled oscillator is set to 24 MHz by using the macro expression "CS_CTL0_DCORSEL_4". The low frequency oscillator is enabled and set to 32 768 Hz (default value), while the high frequency one is set to 48 MHz, both put in maximum drive strength and current consumption mode.

After it is verified that the HFXT oscillator fault flag is cleared, the clock systems frequency are set. MCLK is selected from HFXT to 24 MHz (division with 2), SMCLK is taken also from HFXT and set to 12 MHz (division with 4) and ACLK is taken from LFXT and set to 32 768 Hz (default value).

5.6.3 LED blinking configuration

For this feature, in order to know for sure that the module was correctly initialised, an LED will blink when the LaunchPad is powered up, if both the module and the tag are configured. For this task, the red LED, also used for the alarm, will be used. Another source file "led.c" and a header "led.h" are created. This time, the LED has only debugging purposes.

First step is the initialisation of the port P1.0 as an output one, in the exact same manner as done in 5.2.1. The three registers are set and then we must initialise the LED on logic level 0 to make sure that it is turned off when a power source is connected.

Now, all that remains is to make the LED blink. For the alarm, that operation was done using two separate functions to turn the LED on and off at different times, using a timer interrupt. This time, because the frequency of the signal is not that important, the blinking will be done by toggling the LED. This means that instead of seting the port on 1 or 0 using AND/OR operators, we will

make use of the XOR operator, which toggles the port continuously between the two logic levels. This is done in a void function called “led_blink()”.

5.6.4 Timer configuration for RFID

For the Radio Frequency Identification feature, timer A1 will be used and configured using the “timer.c” source file. As in the case of the alarm, two or three functions will be needed, depending on the users preferences on how to stop the timer. Before, it was needed a separate function to stop the timer, as that was a separate event that happened when the button interrupt took place. Now, this is not necessarily to be done, so the timer can be stopped only inside its working function.

In the void function “timerA1_init(void)”, after the control register was set to stop mode, SMCLK is selected as the clock source and its frequency is divided by 4. Then the timer is reseted using the clear field from the register and finally, the interrupt is disabled from the capture and control register.

The turning on of the timer will be done a little bit different than for the alarm, by using a void function (“start_timerA1”) with an integer parameter called “time”. This method allows the user to easily modify the interrupt period of the timer, by calling the function with different values as parameters. First thing to do is to clear the interrupt flag. Then, the Capture and Compare Register (which stores the values to which the timer has to count) will be written with a value equal to “time” multiplied with 3. The timer is set in up mode, meaning that it will count from 0 to the value stored in the CCR0 register. If there are no interrupt pendings, the timer is stopped by writing the hexadecimal value 0xFFCF in the Control Register (to avoid overwriting).

5.6.5 UART configuration for RFID

Because in this application, some data also needs to be displayed on the serial monitor, another UART interface will be configured in the source file “uartInterface.c”. The process is exactly the same as for the GPS uart interface, the only difference is that now, the baud rate will be set to 9600, instead of 115 200.

The first function would be for port initialisation and is identical to the function described in 5.4.3., as also P1.2 and P1.3 are used and is called “initUartPorts()”.

Next, to set up the working baud rate we will use a function called “initUartInterface()”. The process is the same as for the GPS UART, first time the software reset is enabled and then SMCLK is selected as clock source. The difference is that this time, SMCLK frequency is 12 MHz instead of 3.

After that, the calculus for the baud rate must be done. The frequency divided by the baud rate multiplied with 16 gives the result 78.125. That means that the Baud Rate Control Word Register will store the integer part of the result (78). The Modulation Control Word Register will store the decimal part, a value from Table 24-4 in the Reference Manual [28] that is closest to the given result. Also, in Table 24-5, this case with working frequency 12 MHz and baud rate 9600 is already given by default. In this way, the Reference Manual helps the user to set up all the registers as needed. By knowing this, the Modulation Control Word Register is written with the value 0x0021 in hexadecimal.

Last, the Uart is initialised by disableing the software reset and clearing the interrupt flag register.

5.6.6 SPI configuration for RFID

The Serial Peripheral Interface is the protocol used for communication between the RFID and the MCU, not used in this project until this moment. For its configuration, three pins are needed, one for the clock, one for MOSI (master out, slave in) and one for MISO (master in, slave out).

In the source file called “spi_RFID.c”, the first function created (“void spi_PortInit()”) has the purpose to initialise the needed ports. Port P1.4 wil be used for the clock, P1.6 for MOSI and P1.7 for MISO. Looking into the datasheet of the LaunchPad, we observe that for this configuration, the Select 0 register must be on logic level 1 and Select 1 on 0 for all three ports.

But for this protocol, another two ports are needed, one for the Slave Select (P6.5) and one for enable (P6.4). Before configurint them, we must assure that the enable pin is on the low state. After looking in the datasheet, both Select 0 and Select 1 registers were set on 0, while the Direction register was set on 1, for both ports. For the last operation in this function, the Slave Select pin was initialised with 1, which means that the interface is stopped for the moment.

Next function is a simple one and has the purpose to enable the RFID, by simply changing the output of the enable port (P6.4) to high state. The function is called “enable_TRF7970A()”.

The next function is the hardest one in the configuration of the interface and is called “spi_config()”. The first task that is done, same as in the case of the UART interface, is to enable the software reset from the Control register. The registers for the SPI are accessed with the expression “EUSCI_B0_SPI”. After that, a couple of operations needs to be done in the same register. The direction of the shift register is chosen to start with MSB, then select the master mode and the 3 pin configuration. Last thing, the synchronous mode is enabled and SMCLK is chosen as clock source. The clock phase is set to 0 and clock polarity too.

This register allows the user to choose the character length between 7 and 8 bit. For this application, the 8 bit mode was selected, by placing 0 in the character length field.

Following task is to set the bit rate from the Bit Rate Control Register in order to obtain a working frequency of 4 MHz for the SPI. For this, the value 0x0003 is introduced in the register. To complete the configuration, the software reset is disabled.

Two void functions are created to start and stop the transmision of the RFID (“start_SPI_RFID_transmision” and “stop_SPI_RFID_transmision”). As said before, the transmision is controlled by the Slave Select port, which, opposite than the enable pins, starts the communication when it is on 0 and stops when on 1.

Next functions required are for writing and reading from the RFID. The writing function (“write_SPI_RFID”) uses an integer parameter called TX. A while loop is used to wait until the transmision buffer is empty. If there is no transmit interrupt pending, the content of the transmit buffer will move to the TX variable. A second while waits until data is sent to the receiver, then the timer is turned on using the “start_TimerA1” function with parameter 1. The reading function (“write_SPI_RFID”) has no parameter, but is of type int. If there is no transmit interrupt pending, the transmit buffer is filled with zeros, and if no receiver interrupt is pending, the receiver buffer is returned by the function.

The last function is called “rawwrite_SPI_RFID” and is different from the normal write, because it is used to write a buffer of commands. It uses as parameters a pointer to the buffer and the length of it. Until the full buffer is covered, the transmit buffer places tha data to the memory location of the parameter and with every operation, the index is incremented.

5.6.7 **RFID configuration**

The most difficult part for this feature follows now and is done in a source file called “rfid.c” and consists of 11 very important functions. We start with the less complex function, used to send a command (“command_RFID”) and that uses a parameter (“int command”) that stores the numeric value of the command. The transmision is first turned on, then the “write_SPI_RFID()” function is used to write to command in the memory of the FIFO (first in, first out) register at the adress 0x1F. The process stops when TX is complete and the read value is 0x80.

Next function (“write_RFID_reg”) is used to write a command in a register and uses two parameters for the address of the register and the value of it. After turning on the transmision, the write function is used to locate the address of the register and then tell the value to be placed in it. After that the transmision is turned off. To turn on the radio frequencies, this function is used, so

that in the chip status register (0x00), the value 0x20 is sent (command which turns it on). This operation is done in another function called “turnRF_on()” [29].

Data from the RFID can be read in two modes: single or continuously, for which two different int functions with an int parameter “reg” were created. For the single read, a variable of 16 bits called “value” was defined. After the start of the transmision, the register address is read until RX if complete. Then, the “value” parameter is filled with the data read from the RFID and then returned by the function.

For the continous read function, the only difference is that there is no need to start and stop the transmision and the third bit from the adress and command word is changed to 1 (that means continuous mode).

Now, for the configuration of the RFID, two commands must be sent using the “command_RFID” function. One to initialise the software (0x03) and one for an idle direct command (0x00).

The module needs to scan to see if any data comes to it from another external field, to avoid other RFID devices from perturbating the communication (check if the channel is empty), operation done through the “checkExternalField()” function. The receiver is activated with the value 0x02 sent to the chip status control and the search for radio frequencies start by sending the command 0x19. After a delay, the data read from the RFID is stored in a variable that is later compared to 1. If it is bigger, the function returns 1, otherwise it returns 0.

The initialisation of the RFID module is done inside an int type function “rfid_InitModule()”. There we first make use of the LED previously configured. In the beginning, the blink function of the led is called and then the RFID is reseted. It is checked if there exists RF communication using the above described function, and if yes, the function returns 0. Otherwise, the RF is turned on and the Iso Control Register (0x01) is written with 0x02 value. The value 0x01 is written in the Modulator Control Register (0x09) and the RX No Response Wait Time Register (0x07) is written with the value 0x15. The blinking function is called again and the function will then return 1 [29].

“singleSlotInventory()” is a complex function used to store the received data in an inventory. We create a buffer of length 8 that will store more command values. In the beginning of the function, the raw function is called to write all the values in the buffer (reset FIFO, Send with CRC, write continuous, ISO15693 flags, inventory command mode, mask length). After a delay of 10 miliseconds, the Interrupt Request Register is read continuously from and data is stored on the first position of a vector. This process is done a second time and the result is stored on the second position and finally, the FIFO Status Register is read from and data stored on the last position. Between all these processes, the transmision must be turned on and off everytime and a dummy read is also required. The function returns the vector with the data.

The next task that is required is to read the unique identifier of a tag. This is done inside the “rfid_TagUID()” function, which begins with the declaration of a variable “i” and the blink of the LED. The result of the “singleSlotInventory()” function is stored in a variable. If “i” is positive, then the transmision can begin. A matrix will store the data read from the FIFO register (address 0x1F). As long as “i” is still greater than 0, the receiver buffer will be emptied and the unique identifier will be found in the resulting matrix.

The last function is called ”resetRFID()” and it does the next commands. First it does a software initialization (0x03) followed by an idel command and the start of the timer. Next, the FIFO is reseted (0x0F) and the Modulator Control and Regulator Control Registers are written.

5.6.8 **RFID functionality**

After including all the header files needed, in the main source file we declare a matrix called “tagID[16][10]”, that will store the unique identifier and a pointer to a string, called “slotInventory” to store the data from the FIFO register. An index is required, so it is declared and initialised with 0. First thing to do is to call all the functions for port initialisation (led, spi, uart, clocksystem), followed by the configuration functions. The module is then enabled.

Inside a while(1) loop takes place the whole reading process. The condition for proper functionality is for the module to be correctly initialised, meaning that the proper function must return 0. In that case, the tag function is called and the index is equalised with 9. The unique identifier of a tag is on 8 bytes, the first two positions being for the communication flags. So because of that, the index will cover positions from 2 to 9. As long as the index is still greater or equal with 2, if there is no transmit interrupt pending, the UART transmit buffer will get the tag ID. The index is then decremented and the next character is going to be sent throught the UART to the serial monitor.

6 Experimental Results

Each of the components used in this project were first tested separately from the main code, to observe their working. Unfortunately, there are some barriers that stops the user from properly showing the experimental results of the project.

First of all, Code Composer Studio does not have a serial plotter option to observe the exact manner signals work on every port. Secondly, because of the previous problem, the functionality of the buzzer can not be demonstrated. Only because it has a similar configuration to the LEDs, the fact they are both working in tandem can be deduced. The turning on and off of both devices can not be observed from pictures, only by using an oscilloscope, which I do not own.

As the alarm triggers when an impact of 2 G is detected, this testing can be very hard and unsafe to try at home. But as said above, all the components were tested separately before implementing their code in the main file. The example codes offered by Texas Instruments could be of great use, but as there was no similar application in Resource Explorer and the alarm part was the simplest one, the code used for testing is the same as the final one used in the source file.

Starting with the LEDs, they need to blink every second. They must be on while the buzzer is beeping, and off while the buzzer is silent. Their functionality can be observed in Figure 6.1, which demonstrates that they are both turned on at the same time (the LCD is on just because it is connected to the power supply).

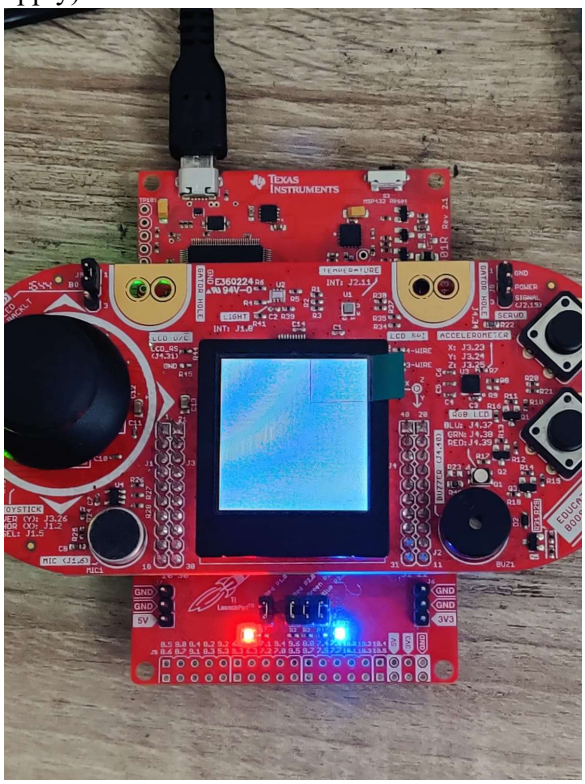


Figure 6.1 Alarm LEDs turning on

Even though the triggering of the alarm could not be home tested due to limitations appeared from the wire connection of the LaunchPad to the laptop, the code could easily be modified in order to observe the proper working. If the timer start function is placed anywhere else in the working loop and is not limited by the condition from the accelerometer, the alarm will work exactly as we wished for, but continously.

In the exact same manner, the push-button was verified to work properly. Although the timer start function was placed somewhere else in the code, the condition for the push-button still remains. So its testing was done while the alarm was working continuously. It was observed that when the button was pushed, the alarm stopped, exactly how we wanted to happen

The accelerometer had a lot of examples that could be tested with, but in order to make it master, an example from Resource Explorer was used, to see the proper working and results given by the aceelerometer on the serial monitor. The results can be seen in Figure 6.2.

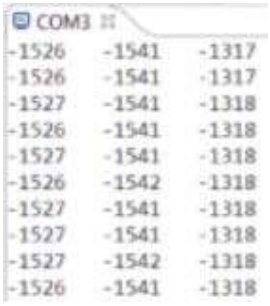


Figure 6.2 Accelerometer result

The first column contains the data for the x-axis, the middle one is for the y-axis and the column from the right is for the z-axis. It was observed how when moving the LaunchPad in different directions, the result change their values during the movement and then return to these static result.

Also, as the movement was faster, the variation of the result was bigger. Due to the LaunchPad being connected to the laptop through an USB cable, bigger acceleration was impossible to be tested, as the device was not yet implemented to be wireless.

The data from the accelerometer is not displayed in the proper application, because they are of no use to the dog owner and the microcontroller would be busy with sending data to the UART continuously for two devices, instead of just one.

The GPS presented one of the hardest parts of the project. First time, an example downloaded from the website where the module was bought from was tested in Arduino. The testing was pretty difficult as the module needed about 30 minutes to find the to communicate with. The GPS has a blue (or red) LED that starts blinking when satellites are found. For a proper use and positioning, it is recommended that the module is tested outside or at least in the balcony or near an open window, for the signal to be stronger.

When the GPS can not find any satellites for communication, the LED is turned off and the result printed on the terminal would be equal to 0, as can be seen in Figure 6.3.

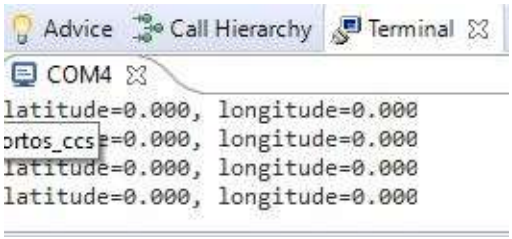


Figure 6.3 Results when GPS does not find any satellites

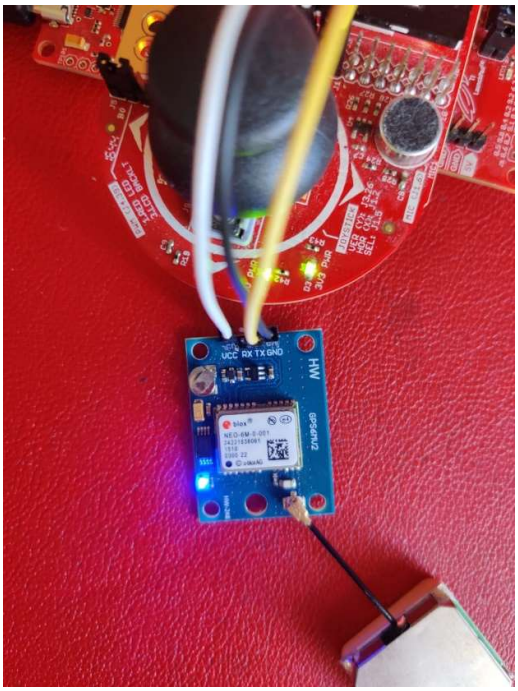


Figure 6.4 LED blinking after the GPS connected to satellites

There is no problem, theoretically, if the GPS module loses connection with the satellites. The program will still run and display null values and the next time it finds connection, the correct position will be displayed, without any extra errors caused by deconnection.

After the LED starts to blink (Figure 6.4), the application may start. The results were printed in the terminal (Figure 6.5). To make sure that everything works properly, the given data was written in Google Maps, to see if the given location is the right one. The error was less than 30m, which is due to the small number of digits the latitude and longitude was displayed (only 3 decimals). For a more accurate positioning, more decimals should be used (around 6 for an error less than 1 meter).

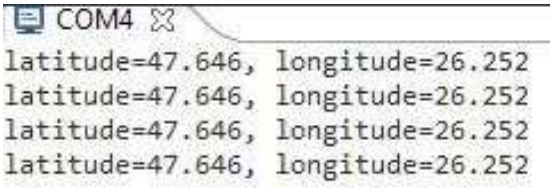


Figure 6.5 Result when the GPS finds satellites

As for the RFID, the thing that we expect to happen is to observe if the tag is recognized and if its unique identifier is printed. First of all, when loading the code on the board, the red LED which was also used in the alarm must blink, demonstrating that the module is correct initialised. If that happens (Figure 6.6), we can proceed further and see what information is displayed on the serial monitor. As wished for, when nothing is scanned, a string of 16 bytes full of zeros is shown. After we place the tag near the RFID scanner, a change in the string can be observed. Now, the unique identifier of the tag is displayed, as can be seen in Figure 6.7.

The implementation of the RFID was built to only recognize and read the identifier of the tag used. A more complex software implementation would be needed to stop writing the identifier once it was scanned and continue to display zero values until a new tag is scanned.

Figure 6.7 Results from the RFID tag scan

The range from which the tag can be scanned is aproximately 8 centimeters. The RFID technology works at far more bigger ranges, but the BoosterPack is a cheap one, for small applications usage only. This is the reason why the range is limited for this project, but still, there is no need for a bigger range. The tag will be placed on the collar of the dog and the scanner on the bowl. This means that when going to eat, the dog`s neck approaches very close to the scanner. So the range will not be a problem.

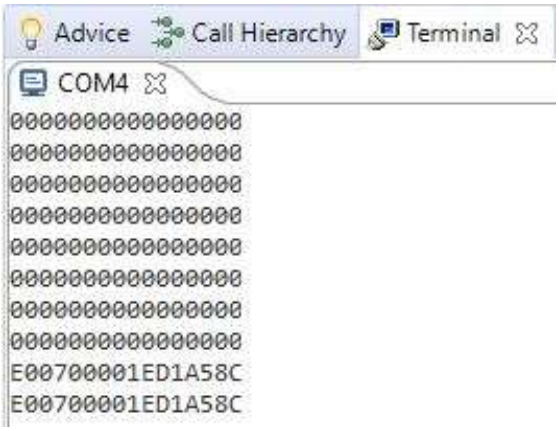


Figure 6.6 LED for RFID blinking

.

7 Conclusions

This project's purpose is to implement the concept of Smart Dog Collar using more simple features. For a simple implementation, a LaunchPad and two BoosterPacks from Texas Instruments were used. Due to their big size and their need of connection to a laptop through an USB cable, the project is not yet ready to be placed on a wireless collar, at the neck of a dog.

Also, as the boards are not protected by anything, it would be a high risk to even keep it outside. Not only a rain would damage the collar, but also a dog playing around could easily destroy it.

As a dog owner for more than 12 years, I always found myself scared when my dog ran away. Also, one of my previous dogs was very unhealthy and did not eat for two days straight, without me or my family noticing, while it had a very serious disease. You never know when two days can make a difference so it is always better to be safe than sorry. This was one of the primary motivations to choose this theme and implement it for my bachelor degree.

But as for the concept stage, the Smart Dog Collar I implemented gets its job done, exactly as expected. The alarm works perfectly, one second on and one second off. The accelerometer used for impact detection reads data continuously, without displaying them, as there is no need for a dog owner to measure this. Even if the accelerometer can not yet be tested as an impact detector due to previously mentioned reason, it can also be used in more features for the upgrade that I already have in mind.

The GPS was a pretty difficult component to use at first, but it will come in handy after first making it work. It sends the exact position with a very small error (the actual error can be reduced from the code, there is no need of a more expensive device) and the result could be verified on Google Maps. As for its fiability, this device will also not be good for using on a dog, as its size is way too big.

But for future upgrades of the collar, the GPS might give many more information than it does now, in the actual stage. For example, it could get the speed of the dog and its direction of movement. The GPS can further be connected to a GPS module or to the internet, so that the owner could see the position of the dog live, on the map, right on their phone.

The RFID module looks very promising, this feature will be very interesting after implementing a system to let only an amount of food to fall into the bowl. Right now, it only recognises and gets the identifier of the tag, without creating eny event when this action occurs. This mai present a new beginning in the nutrition of the dogs, allowing the owners to keep an exact count of the eaten food. This will be a very good benefit for dog trainers, who need to be in a great shape and follow a strict diet.

I really enjoyed documenting and working for this project, even though I found it pretty hard in the GPS and RFID parts. I am looking forward to further continue and develop more features for the Smart Dog Collar, including an ergonomic and much smaller version. Adding to that, I hope that after my masters degree I will have the knowledge to create a mobile applications for the Smart Dog Collar, which I now see as the most developed version in my mind.

8 References

[1] "PetSafe anti barking collar." https://abc-zoo.ro/22947-petsafe-anti-barking-collar-rechargeable-lite.html?gclid=CjwKCAjwtdeFBhBAEiwAKOIy58zVIamTFf_fJj1YiGrw_gIPYaZAYtCQVIFDnXoqovDhn6WdDxwONBoC4GIQAvD_BwE (accessed Jun. 01, 2021).

[2] "PetSafe Smart Dog training collar." https://www.electric-collars.eu/training-collars/petsafe-smart-dog-training-collar?gclid=CjwKCAjwtdeFBhBAEiwAKOIy54SaHIztmEYvP_BeSd8fYC3AOAHt5-jRGcNQleGDzO2x8wj2gI7oXhoCMnwQAvD_BwE (accessed Jun. 01, 2021).

[3] "Microchipping: What is it and why should you do it?" https://www.softpaws.com/microchipping-what-is-it-and-why-should-you-do-it/ (accessed Jun. 02, 2021).

[4] "GPS Dog Collar Pet Tracking Device." https://lcpshop.net/product/gps-dog-collar-pet-tracking-device/?gclid=CjwKCAjwtdeFBhBAEiwAKOIy54V4CxIlwwDWqVMm1roNnrK5ddk0XJkKdi3L2aDkB-CfPN5297OjqRoC03MQAvD_BwE (accessed Jun. 01, 2021).

[5] "LED Light Spectrum Enhancement with Transparent Pigmented Glazes," 2016. https://www.led-professional.com/resources-1/articles/led-light-spectrum-enhancement-with-transparent-pigmented-glazes-by-light-spectrum-glazes (accessed May 26, 2021).

[6] "Adafruit - Using Piezo Buzzers with CircuitPython & Arduino," 2018. https://learn.adafruit.com/using-piezo-buzzers-with-circuitpython-arduino (accessed May 23, 2021).

[7] "Physics of Music-Notes." https://pages.mtu.edu/~suits/notefreqs.html (accessed May 24, 2021).

[8] "Khan Academy - What is acceleration?" https://www.khanacademy.org/science/physics/one-dimensional-motion/acceleration-tutorial/a/acceleration-article (accessed Jun. 08, 2021).

[9] "Acceleration," *The Gale Encyclopedia of science*. 2018, [Online]. Available: https://www.encyclopedia.com/science-and-technology/physics/physics/acceleration#A.

[10] A. Venosa, "Breaking Point: What's the Strogest G-Force Humans Can Tolerate?," 2016. https://www.medicaldaily.com/breaking-point-whats-strongest-g-force-humans-can-tolerate-369246 (accessed Jun. 11, 2021).

[11] E. L. C. Shepard *et al.*, "Identification of animal movement patterns using tri-axial accelerometry," *Endanger. Species Res.*, vol. 10, no. 1, pp. 47–60, 2010, doi: 10.3354/esr00084.

[12] "Compass Sensor – Working and Applications." https://www.elprocus.com/compass-sensor-working-and-applications/ (accessed Jun. 12, 2021).

[13] E. Geier, "5 Frightenly Common Reasons Dogs Run Away." https://www.rover.com/blog/why-dogs-run-away/ (accessed Jun. 12, 2021).

[14] J.-M. Zogg, "GPS basics," *U-Blox, Thalwil*, vol. 1, no. January, pp. 1–60, 2002, [Online]. Available: http://w3.leica-geosystems.com/downloads123/zz/gps/gps_system500/manuals/GPSBasics_de.pdf%5Cnhttp://lab.sgg.whu.edu.cn/gnss/docs/GPS_basics_u_blox_en.pdf%0Awww.u-blox.com Page.

[15] "How GPS Started." https://www.fleetistics.com/resources/gps-history-benefits/ (accessed Jun. 12, 2021).

[16] T. Wood, "Who owns our orbit: Just how many satellites are there in space?," 2020. https://www.weforum.org/agenda/2020/10/visualizing-easrth-satellites-sapce-spacex (accessed Jun. 12, 2021).

[17] M. Sullivan, “A brief history of GPS,” 2012. https://www.pcworld.com/article/2000276/a-brief-history-of-gps.html (accessed Jun. 12, 2021).
[18] J. P. Leite, “A brief History of GPS in Car Navigation,” 2018. https://ndrive.com/brief-history-gps-car-navigation/ (accessed Jun. 12, 2021).
[19] G. Blewitt, “Basics of the GPS Technique : Observation Equations §,” *Geod. Appl. GPS*, pp. 1–46, 1997.
[20] “Basic Calorie Calculator.” https://vet.osu.edu/vmc/companion/our-services/nutrition-support-service/basic-calorie-calculator (accessed Jun. 14, 2021).
[21] S. Ahuja and P. Potti, “An Introduction to RFID Technology,” *Commun. Netw.*, vol. 02, no. 03, pp. 183–186, 2010, doi: 10.4236/cn.2010.23026.
[22] E. Ilie-zudor, Z. Kemény, P. Egri, and L. Monostori, “the Rfid Technology and Its Current Applications,” pp. 29–36, 2006, [Online]. Available: http://igor.xen.emi.sztaki.hu/~zudor/Publications/RFID_technology and applications.pdf.
[23] P. Dhaker, “Introduction to SPI Interface,” *Analog Dialogue*, no. September, pp. 1–5, 2018, [Online]. Available: https://www.analog.com/media/en/analog-dialogue/volume-52/number-3/introduction-to-spi-interface.pdf.
[24] Texas Instruments, “Development Kit (MSP - EXP432P401R),” 2018.
[25] “ENERGYTRACE.” https://www.ti.com/tool/ENERGYTRACE?DCMP=ep-mcu-msp&HQS=energytrace (accessed Jun. 15, 2021).
[26] Texas Instruments, “Msp432P401R, Msp432P401M,” 2015. [Online]. Available: http://www.ti.com/lit/ds/symlink/msp432p401r.pdf.
[27] Texas Instruments, “BOOSTXL-EDUMKII Educational BoosterPack™ Plug-in Module Mark II,” 2015. [Online]. Available: http://www.ti.com/lit/ug/slau599a/slau599a.pdf.
[28] Texas Instruments, “Technical Reference Manual.”
[29] Texas Instruments, “MULTI-PROTOCOL FULLY INTEGRATED 13 . 56-MHz RFID / NEAR FIELD COMMUNICATION (NFC) TRANSCEIVER IC TRF7970A,” 2012.

Razvan-Gabriel Prepeliuc

DATA NAȘTERII:
31/07/1998

CONTACT

Cetățenie: română

Gen: Masculin

Marasti 1, null
720182 Suceava, România

razvan.prep31@gmail.com

(+40) 745034575

EXPERIENȚA PROFESIONALĂ

19/08/2019 – 25/09/2019 – Jucu, România

Working Student

Robert Bosch SRL

EDUCAȚIE ȘI FORMARE PROFESIONALĂ

15/09/2013 – 23/05/2017 – Str. Mihai Viteazu nr. 24, Suceava, România

Absolvent

Colegiul National "Petru Rares" Suceava

http://cnprsv.ro/

01/10/2017 – 20/07/2021 – Str. George Barițiu nr. 26-28, Cluj-Napoca, România

Licentiat

Universitatea Tehnica Cluj-Napoca, Facultatea de Electronica, Telecomunicatii si Tehnologia Informa

https://etti.utcluj.ro/

COMPETENȚE LINGVISTICE

LIMBĂ(I) MATERNĂ(E): română

ALTĂ LIMBĂ (ALTE LIMBI):

engleză

Comprehensiune orală	Citit	Exprimare scrisă	Conversație	Scris
C1	C1	C1	C1	C1

COMPETENȚE DIGITALE

Navigare Internet / Social Media / Utilizare buna a programelor de comunicare(mail messenger skype) / Microsoft PowerPoint / Microsoft Word / Zoom / Google Drive / Android / Internet Explorer Edge Mozilla Google Chrome / C programming / Adobe

COMPETENȚE DE COMUNICARE ȘI INTERPERSONALE

Competențe de comunicare și interpersonale

- abilitati de comunicare foarte bune dobandite in urma prezentei in doua proiecte europene, organizate de Academia Europeana Otzenhausen in 2016 si 2017 si dezvoltate in cadrul a mai multe trupe de muzica din care am facut parte

PERMIS DE CONDUCERE

Permis de conducere: AM

Permis de conducere: B1

Permis de conducere: B