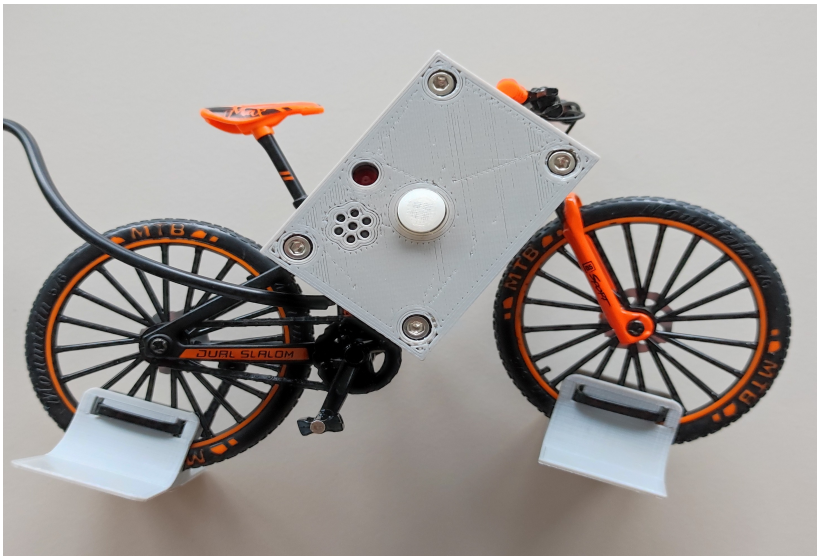
Entwicklerdokumentation

A tiny bike sentry

WS 2024, Projekt Eingebettete Software

Lukas Krämer



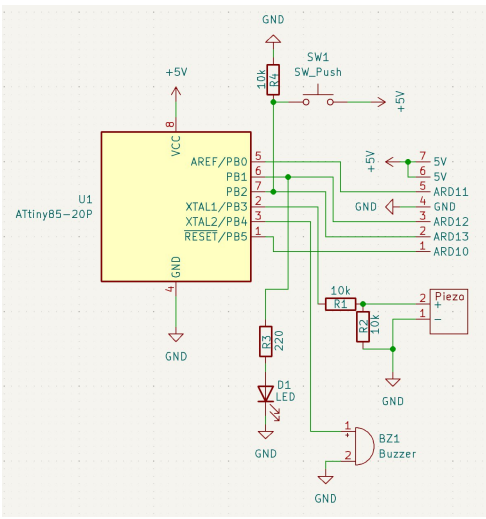
Einführung

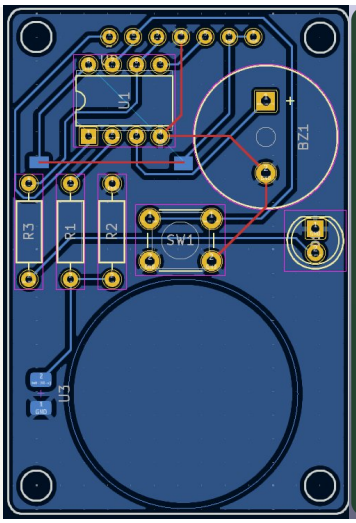
Fahrraddiebstahl stellte mit über 250.000 Delikten im vergangenen Jahr (Quelle 1) eine der weit verbreitetsten Straftaten in Deutschland dar, die aktuellen Präventivmaßnahmen gegen Diebstahl des eigenen Fahrrads reichen dabei von klassischen Schlössern bis hin zu Fahrrad-Sammelschließanlagen, wie sie sich beispielsweise an einigen Bahnstationen finden lassen. Gemeinsam haben dabei beide Lösungen, dass sie die Wahrscheinlichkeit des Diebstahls zwar erheblich senken, dafür aber auch verhältnismäßig unpraktisch und zum Teil auch mit hohen Kosten bzw. Abomodellen verbunden sind. Oftmals würde allerdings ein einfaches akustisches Warnsignal beim unerlaubten Bewegen des Fahrrads reichen, um einen potentiellen Diebstahl zu verhindern. Insbesondere in solchen Fällen, in denen ein Fahrrad zwecks Bequemlichkeit gar nicht gegen Diebstahl gesichert wird. Die bewährten Maßnahmen, wie beispielsweise das Anschließen, könnten und sollten natürlich weiterhin genutzt werden.
In diesem Projekt wurde ein solches rein akustisches Fahrradschloss entwickelt, dass vom Fahrradbesitzer beim Hinterlassen seines Fahrrads manuell aktiviert werden muss und anschließend mittels eines Erschütterungssensors (Piezoelement) kontinuierlich prüft, ob eine Erschütterung und somit ein potentieller Diebstahl vorliegt. Ist dies der Fall, gibt das Gerät sowohl optische als auch akustische Warnsignale von sich, um den Dieb doch noch von der Straftat abzuhalten. Sollte der Alarm fälschlicherweise aktiviert werden (zum Beispiel aufgrund einer kräftigen Windböe), schaltet er sich nach einer gewissen Zeit ohne weitere Erschütterungen automatisch wieder ab, bis das Gerät erneut bewegt wird.

Hardware

Neben der Software, auf welcher der Fokus bei diesem Projekt lag und auf die an späterer Stelle noch näher eingegangen wird, bestand der Gesamtaufwand nichtsdestoweniger auch aus einem Hardwareteil.
Zum Einsatz für das eigentliche Fahrradschloss kam hier ein ATtiny85, der im Vergleich zu anderen Mikrocontrollern eine sehr geringe Stromaufnahme und kleinen Platzbedarf besitzt und sich daher ideal für das Projekt eignet, da es langfristig mittels Akkus oder Batterien betrieben werden soll. Zwar verfügt der Controller verglichen mit beispielsweise einem Arduino Nano oder Raspberry Pi Pico über wenig digitale Ein- und Ausgänge, Speicher und Rechenleistung, die aber für dieses kleine Projekt dennoch vollkommen ausreichend sind.
Für die Erkennung der Erschütterungen wurde ein Piezoelement verwendet, das unter mechanischer Verformung hohe Spannungsspitzen erzeugt, dabei aber erheblich weniger Strom (genauer gesagt: praktisch gar keinen!) benötigt als herkömmliche Beschleunigungssensoren mit I2C Schnittstelle, die sicherlich auch hätten verwendet werden können. Die Spannungsspitzen des Piezoelements entstehen auch bei leichten Erschütterungen, fallen jedoch entsprechend geringer aus. Für das Auslesen des Sensors wurde er an einen analogen Eingang des ATtinys angeschlossen, um die Eingangspins vor eventuell hohen Spannungen/Strömen zu schützen, wurde

sicherheitshalber noch ein hoher 10k Ohm Widerstand in Reihe und parallel geschaltet (siehe Bild 1).
Zur Bedienung des Schlosses wird zudem noch ein klassischer Taster benötigt, der an 5V und mittels Pull-down Widerstand an einen digitalen Eingangspin des Mikrocontrollers angeschlossen wurde. Auf die eingebauten Pull-up Widerstände des Mikrocontrollers wurde hier bewusst verzichtet, da diese einen permanenten Stromfluss bedeutet hätten, was die gesamte Leistungsaufnahme der Schaltung im Standby Zustand unnötig erhöht hätte.
Für den tatsächlichen Alarm kommen außerdem noch eine rote LED und ein aktiver Buzzer zum Einsatz, die unabhängig voneinander über den ATtiny geschaltet werden können.
Da der ATtiny nicht direkt über USB programmiert werden kann, wurde auf der Platine für das Schloss noch ein Steckverbinder montiert, der eine ISP-Schnittstelle zu einem anderen Mikrocontroller bereitstellt. Als ISP-Programmer wurde ein Arduino Nano Every verwendet, den Schaltplan zu der Programmer-Platine ist in Bild xxx dargestellt.





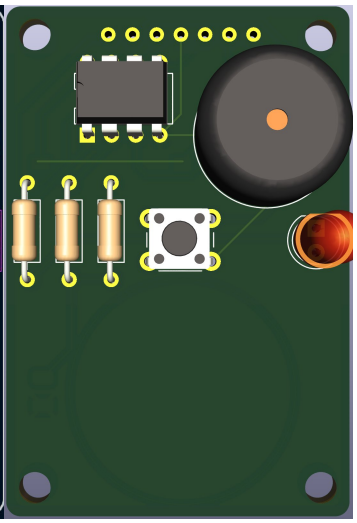


Bild 1: Schaltplan *Bild 2: PCB Layout* *Bild3: PCB*

Software

Ganz im Sinne des Modulnames „Eingebettete Software“ lag der Hauptfokus beim „Bike Sentry“ auf einer modularen und durchdachten Softwarearchitektur, die flexibel bezüglich späteren Veränderungen oder Ergänzungen sein sollte und dennoch gut lesbar, wartbar und strukturiert ist.
Zur Entwicklung wurde Visual Studio Code in Kombination mit dem Kommandozeilen-Tool Arduino CLI benutzt, um den geschriebenen Code für den ATtiny zu kompilieren und im Anschluss über den ISP-Programmer hochzuladen.

Wesentlicher Bestandteil des Codes ist die „main.ino“ Datei, in der die gesamte Programmlogik abgebildet ist. Hauptsächlich wird hier mit einem Aufzählungstyp (enum) gearbeitet, der die folgenden Zustände abspeichern kann:

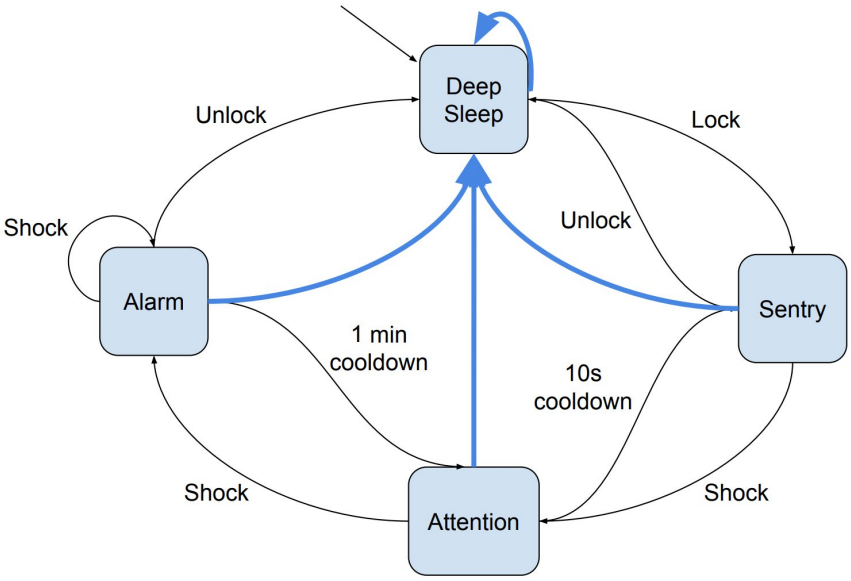


Bild 4: Zustandsgraph

Wie oben dargestellt, kann sich das Fahrradschloss in vier verschiedenen Zuständen befinden, zwischen denen teils manuell durch den Nutzer (Lock, Unlock per Tasterdruck), teils aber automatisch bei Erschütterungen (Shock) oder nach einer gewissen Zeit ohne weitere Erschütterungen (Cooldown) geschaltet wird. Nach dem Einschalten befindet sich das Gerät zunächst im Zustand „Deep Sleep“, in dem sich der ATtiny im Schlafmodus befindet und per Interrupt auf einen Tasterdruck wartet, was ihn in den Zustand „Sentry“ befördert. In diesem Zustand liest der Mikrocontroller permanent das Piezoelement aus, um potentielle Erschütterungen erkennen zu können. Falls dies der Fall ist, wird der Alarm allerdings noch nicht direkt gestartet, sondern das Gerät gelang zunächst in einen Zwischenzustand „Attention“ und löst erst nach einer weiteren Erschütterungen innerhalb der nächsten zehn Sekunden durch einen Wechsel in „Alarm“ den eigentlichen Alarm aus. Falls in dieser Zeitspanne keine weitere Erschütterung registriert wird, handelt es sich um einen Fehlalarm und es wird zurück in „Sentry“ gewechselt.
Im Alarmzustand gibt das Schloss sowohl optische als auch akustische Signale von sich, um auf den Diebstahl aufmerksam zu machen. Abgebrochen wird der Alarm nach einer Minute automatisch, andernfalls kann er auch jederzeit durch einen Knopfdruck manuell terminiert werden, da dieser einen Wechsel in den Zustand „Deep Sleep“ zur Folge hat. Dieses Verhalten, also dass ein Knopfdruck in anderen Zuständen als „Deep Sleep“ unmittelbar in ebendiesen Zustand wechselt, haben alle Zustände gemeinsam, um nicht zu Verwirrung zu führen und eine einfache Bedienung zu ermöglichen. In anderen Worten: ein Kopfdruck führt immer zu dem vorhersehbaren und gleichem Verhalten, unabhängig vom internen Zustand.

Der Vorteil dieser Programmarchitektur liegt hauptsächlich darin, dass sich jederzeit weitere Zustände ergänzen lassen, da diese in der Hauptschleife einfach mittels

switch-Anweisung abgearbeitet werden. Dadurch war auch die Implementierung vergleichsweise einfach möglich, da zunächst die gesamte switch-Anweisung geschrieben und dann nach und nach das Verhalten in den Zuständen implementiert und separat getestet werden konnte.
Die Hauptdatei behandelt auch alle Formen von externem Input (Piezo und Taster) sowie die Implementierung der Schlafmodi und Interrupts. Diese wurden nicht in andere Dateien ausgelagert, da sie unmittelbaren Lese- und Schreibzugriff auf den globalen Aufzählungstyp für die Zustände benötigen und das Umherreichen von Zeigern auf diesen Typen die Les- und Wartbarkeit des Programms erheblich verschlechtern würde.

Was allerdings ausgelagert ist, sind alle Funktionen, die Ausgänge des ATtinys manipulieren, um die Hardware zumindest in Teilen von der Logik in der main-Datei zu trennen. Dafür wurde zunächst die Schnittstelle für die entsprechenden Funktionen in der Datei „Gpio.h“ festgelegt. Neben primitiven Funktionen wie „on“ und „off“, die jeweils einen übergebenden Pin Ein- bzw. Ausschalten können, gibt es noch „set_pin“ und „toggle“, die einen Pin auf einen Zustand (an, aus) setzen können bzw. einen Pin mit einer Periodendauer für eine bestimmte Anzahl an Vorgängen abwechselnd zwischen ein und aus schalten können.
Die genannten Funktionen sind in der „Gpio.cpp“ Datei nach der entsprechend definierten Schnittstelle implementiert.
Die Gpio Klasse ist dabei universell einsetzbar und kann theoretisch auch für andere Projekte in der Zukunft genutzt werden.

Im Gegensatz dazu steht die Animation Klasse, welche die Gpio Klasse nutzt, um Animationen für das Main-Programm bereitzustellen. Diese sind spezifisch auf dieses Projekt zugeschnitten und dienen dem Zweck, die hardwareseitigen Animationen nicht in „main.ino“ durchführen zu müssen, sondern getrennt von der Logik leicht anpassbare Animationen verwalten zu können. Die Klasse beinhaltet hauptsächlich Methoden, die vom Hauptprogramm beim Zustandswechsel aufgerufen werden können, zum Beispiel: exit_alarm(), welche die Animation beinhaltet, die beim Verlassen des Alarm Zustands abgespielt werden soll. Ähnlich aufgebaute Methoden gibt es auch für die anderen Zustände, mehr Informationen dazu sind in der doxygen Dokumentation zu finden.

Zu guter Letzt fasst die Bibliothek/Klasse „Timing“ angepasste Versionen der delay() und millis() Funktion zusammen. Diese sind nötig, weil der ATtiny hier lediglich mit einem MHz betrieben wird, um Strom zu sparen, die Standard-Taktfrequenz allerdings acht MHz beträgt. Daher wäre der gewöhnliche Aufruf von delay() oder millis() um Faktor acht falsch, was durch „Timing“ basierend auf der aktuell gewählten Taktfrequenz korrigiert wird. Diese kann dabei gemeinsam mit vielen anderen Einstellungen, wie genutzten Pins für Led, Buzzer oder Piezo, Schwellenwerten und Cooldown-Einstellungen, in der Datei „defines.h“ definiert werden.

Neben einer durchdachten Softwarearchitektur sollte auch ein Schlafmodus implementiert werden, dieser wurde hier mittels der der AVR sleep Bibliothek realisiert. Bevor der ATtiny in den „SLEEP_MODE_PWR_DOWN“ wechselt, werden der Analog zu Digital Konverter und Analog Komparator ausgeschaltet, um noch mehr Strom zu sparen. Dadurch konnte die Stromaufnahme im Schlafmodus auf 6,2µA reduziert werden, was selbst bei kleinen Akkus mit nur wenigen hundert mAh Kapazität für mehrere Jahre Akkulaufzeit ausreichen sollte. Aufwachen aus dem Schlafmodus erfolgt durch einen Pin-Interrupt durch den vom Nutzer gedrückten Taster.

Verbesserungspotential

Hardwareseitig sind soweit keine Probleme bekannt, allerdings wäre dauerhaft wünschenswert, wenn das Gerät auch portabel per Akku betrieben werden könnte. Dafür wäre beispielsweise ein kleiner (300mAh) LiPo mit passender Ladeelektronik (z.B. TP4056 oder TP5100) geeignet. Zudem kann der ATtiny auch mit Spannungen kleineren Spannungen als 5V betrieben werden (minimal 1,8V), was die Stromaufnahme um Faktor drei bis vier reduzieren würde. Allerdings würde die Platine dennoch eine zusätzliche 5V Spannungsversorgung benötigen, um im Falle eines Alarms den Buzzer und die LED einschalten zu können.
Darüber hinaus lässt sich der Alarm aktuell einfach per Knopfdruck deaktivieren, was langfristig nicht sonderlich sinnvoll wäre, da dies auch von einem Dieb erfolgen könnte. Gelöst werden könnte dieses Problem durch eine andere Art der Alarmdeaktivierung, wie beispielsweise einen Fingerabdrucksensor oder ein Bluetooth Modul, das die Anwesenheit des Smartphones vom Fahrradbesitzer erkennen kann und das Schloss dann automatisch entriegelt. Um zusätzlichen Hardwareaufwand zu vermeiden, könnte die Pin Eingabe jedoch auch über den bereits vorhandenen Kopf erfolgen. Diese könnte zum Beispiel im Morsecode erfolgen oder sich vollständig frei von Buchstaben und Zahlen konfigurieren lassen, so wäre unter anderem das „Klopfen“ einer Melodie als Entsperrmethode denkbar.

Was die Software betrifft, sind aktuell noch globale Variablen für Zeitstempel vorhanden, um die Animationen korrekt steuern zu können. Dies funktioniert soweit zwar, langfristig wäre es allerdings eleganter, die Zeitstempel intern im Animation Objekt mitzuverwalten.
Auch das Interrupt-Handling sowie die Methoden, um den ATtiny in den Schlafmodus zu versetzen bzw. wieder daraus zu wecken, könnten noch ausgelagert werden. Wie im Teil „Software“ näher beschrieben, würde dies jedoch dazu führen, dass Zeiger zum Aufzählungstypen, der den aktuellen Zustand verwaltet, übergeben werden müssten. Meiner Meinung nach verschlechtert dies die Les- und Wartbarkeit enorm, weshalb ich mich zunächst dagegen entschieden habe. Dadurch wird sichergestellt, dass Lese- und Schreiboperationen auf dem Aufzählungstypen nur im Hauptprogramm und nirgends sonst durchgeführt werden.

Zusätzlich ist die Tasterentprellung zwar soweit funktional, jedoch noch nicht perfekt gelöst, da ich zunächst bewusst auf statische/globale Variablen verzichten wollte, die für eine „richtige“ Tasterentprellung vonnöten wären

Wartung

Da im Projekt keine Servos oder Motoren verwendet wurden, sollte die mechanische Wartung vergleichsweise leicht sein. Die größte Schwachstelle stellen die beiden einzigen beweglichen Teile, der Buzzer und Taster, dar. Diese können im Laufe der Zeit verschmutzen oder sich abnutzen, was im Falle eines Defektes als Erstes kontrolliert werden sollte.
Zudem ist darauf zu achten, den Bike Sentry keinen großen mechanischen Belastungen auszusetzen, da sich die Lötstellen auf der Rückseite der Platine lösen könnten. Dies kann insbesondere nach einem Sturz der Fall sein und dazu führen, dass das Gerät augenscheinlich ganz defekt ist. Hier empfiehlt sich das vorsichtige Öffnen des Gehäuses und kontrollieren aller Lötstellen.

Weiterentwicklung

Entsprechende Weiterentwicklungen können gerne als Pull-Request im folgenden GitHub Repository hinzugefügt werden:
https://github.com/kraemerlukas314/A-tiny-bike-sentry
Auch Feature-Requests oder Bug-Reports können gerne auf gleiche Weise eingereicht werden.

Einrichten der Entwicklungsumgebung

Für die Entwicklung der Software kann prinzipiell ein beliebiger Text Editor genutzt werden. Besonders empfehlenswert ist jedoch die Arduino IDE, da hier der Code anwenderfreundlich auf den ATtiny hochgeladen werden kann. Dazu muss in den Einstellungen diese zusätzliche Board-URL hinzugefügt werden. Anschließend unter „Tools“ -> „Boards“ -> „Boardsmanager“ nach „ATtiny“ suchen und das Paket von „David A. Mellis“ installieren.
Nach einem Neustart der IDE kann nun unter „Tools“ -> „Board“ der ATtiny ausgewählt werden. Nun muss lediglich noch der Prozessor unter „Tools“ auf ATtiny85 gesetzt werden, zudem empfehle ich das Einstellen der Taktrate auf „Internal 1MHz“. Falls eine andere Taktrate gewünscht ist, muss diese außerdem in der Datei „defines.h“ angepasst werden, da ansonsten die delay() und millis() Funktionen nicht wie erwartet funktionieren.
Zum Hochladen muss nun der Arduino ISP Programmer an den PC angeschlossen werden. Falls dieser noch nicht die entsprechende Firmware besitzt, kann diese über „File“ -> „Examples“ -> „ArduinoISP“ -> „ArduinoISP“ auf das Board geflasht werden. Nun kann der ATtiny mit dem ISP Programmer verbunden werden. Damit das

Hochladen funktioniert, ist zunächst der Port unter „Tools“ -> „Port“ korrekt auszuwählen, außerdem muss der Programmer, zu finden unter „Tools“ -> „Programmer“, auf „Arduino as ISP (ATmega32U4)“ gesetzt werden. Falls als ISP Programmer kein Arduino Nano Every, sondern ein Arduino Nano oder Uno verwendet werden möchte, ist die Option entsprechend auf den Eintrag „Arduino as ISP“ (also ohne ATmega32U4) zu setzen. Das Hochladen erfolgt jetzt wie gewohnt über den Upload-Button (Arduino IDE v1) oder über „Sketch“ -> „Upload using Programmer“ (Arduino IDE v2).

Falls auf die Nutzung der Arduino IDE verzichtet werden möchte, kann zum Hochladen auch das im Ordner „src/main“ hinterlegte Upload-Skript genutzt werden. Dieses gibt es sowohl als „.bat“ Version für Windows als auch als „.sh“ Datei für Linux/Unix basierte Systeme.
Hierzu muss allerdings noch Arduino CLI gemeinsam mit der additionalen Board-URL installiert werden, gegebenenfalls muss zusätzlich der Port im Skript selbst verändert werden. Dieser lässt sich unter Windows über den Gerätemanager und unter Linux mittels „ls /dev/tty*“ finden.

Teileliste

- 1x ATtiny85
- 1x aktiver Buzzer
- 1x rote LED
- 1x Taster
- 1x Arduino Nano Every
- Leiterplatte
- 1m dünnes Kabel
- 4x M3 Muttern
- 4x M3 Schrauben
- 30g Filament (z.B. PLA, PETG oder ABS)

Werkzeug

- Lötkolben
- Lötzinn
- 3D Drucker
- M3 Inbus