ESP32-Brevlådemonitor

Emil Kool

Sammanfattning

Brevlådemonitorn var ett 5-veckorsprojekt för att bygga ett system som kan skicka en notis till ett frivilligt redan existerande API när det finns brev i brevlådan. Systemet är tänkt att vara självständigt med en ESP32 och tillhörande elektronik monterad i en brevlåda, en backend enhet som kör 2 docker containrar som anropar ett API för att skicka notiser. För nuvarande så finns det bara ett API som är hårdkodat in men det är upplagt på ett sådant sätt att lägga till ytterligare system och anropa ett eller flera API:er går att göra utan större tidskrav. Jag valde att använda mycket av hårdvaran som jag redan hade införskaffat vilket också medförde några problem i avläsningen om det finns något i brevlådan men som ett koncepttest uppfyllde dom komponenterna målet till en acceptabel nivå. Projektet har lyckats med ett system som skickar notiser när det är relevant men som tidigare nämnt är inte hårdvaran helt anpassat för det här projektet så ibland kommer det felaktiga notiser.

Mål

Målet är att ha ett system som är robust med lång batteritid som kan meddela användaren när någonting finns inuti brevlådan, som ytterligare mål ville jag också hålla koll så att kretsen inne i brevlådan inte har slut på batteri utan att användaren får redan på det. Allt skulle helst kunna monteras i ett litet paket som är lätt att installera i brevlådan utan större kunskaper. Backend sidan ville jag hålla så simpel som möjligt så vem som helst kan köra den så länge dom har möjlighet att köra docker.

Implementation

Det första jag började med var ett simpelt kopplingsschema för all som ESP32 behövde inuti brevlådan. Rent kodmässigt så var det första Wi-Fi och MQTT på ESP32 för att få kommunikationen så tidigt som möjligt för att kunna testa hela kedjan så debugging blir lättare. När MQTT fungerade så satte jag upp 2 docker containrar med docker-compose. Den första är en MQTT broker som kör med mTLS för att säkra kommunikationen. Den andra containern är ett pythonscript som bearbetar meddelandena som skickas för att avgöra om en notis ska skickas.

ESP32

All kod till enheten är byggt med platformIO på espressifs toolchain för att få direkt tillgång till allt som freeRTOS och espressif har att komma med. Jag valde att använda en reed-switch med pull-up resistor för att hålla koll på om brevlådan var öppen eller inte. Elektroniken monterades inne i själva brevlådan medan magneten som stänger reed-switchen sattes i locket (fig 1).

Wi-Fi. All Wi-Fi kod följer espressifs standardimplementation av Wi-Fi enligt deras dokumentation, inklusive att lägga till inställningarna till espressifs konfigurationssystem för att underlätta att migrera till olika nätverk och andra variabler utan att behöva gå in i koden och försöka hitta vart det är definierat.

MQTT. Jag valde att köra MQTT 3.1.1 då jag har tidigare erfarenhet av den versionen till skillnad från det nyare MQTT 5. Då jag har valt att använda mig av mTLS så behövde jag generera certifikat för MQTT servern och alla klienter. Det gjordes med hjälp av openSSL och sedan delades dom ut till enheterna som behöver dom. I konfigurationen innan man kompilerar kan man välja vilken QoS man vill ha på meddelanden.

Avståndsmätare. Modellen av avståndsmätare som jag avvänder mig av är en HC SR-04 ultraljudsavståndsmätare. Avståndsmätaren används för att mäta avståndet mellan kant till kant i brevlådan, ligger något emellan så kommer man att läsa av ett kortare avstånd än

förväntat och då vet vi att någonting ligger där. Kodmässigt var det väldigt lätt att bygga en enkel drivrutin för enheten då den bara har fyra pins och enkel kommunikation av avstånd. Två av pinsen är för positiv volt och jord dom resterande två är för Trigger och Echo. Avståndsmätaren sätter Echo hög tills den får tillbaka ett eko efter man har startat enheten med att sätta Trigger hög. Eftersom vi bara är intresserade av skillnad i avstånd ifrån en tom brevlåda till en brevlåda som har någonting i sig så behöver vi inte bry oss om att konvertera tiden för ett eko till avstånd.

Programflöde. Flödet i hur programmet skulle skicka meddelanden tog några försök att få korrekt. Jag visste tidigt att jag ville få in deep sleep för att kunna spara så mycket batteri som möjligt men var osäker på hur jag skulle väcka enheten. Började med att se hur mycket arbete det skulle vara att använda ULP-Coprocessor men insåg snabbt att det skulle ta för mycket tid för att få någonting inom den korta tidsramen i projektet. Slutade med att jag körde två olika wake sources, RTC interrupt och en tidsbaserad. RTC som wake source var mycket lättare att implementera än ULP och gav mig liknande fördelar, framförallt att det gjorde det väldigt lätt att väcka ESP när brevlådan öppnades. Den tidsbaserade källan är enbart där så att den säger till att den fortfarande kan koppla upp sig för att säkerställa att den fortfarande har batteri och kan komma åt både Wi-Fi och MQTT. Efter den väcks så är flödet enbart där för att bestämma vilket meddelande som ska skickas vidare, det slutade med att ett fåtal steg för att säkerställa att allt var som förväntat. Jag skapade ett flödesdiagram (fig 3) tidigt i utvecklingen för att underlätta för mig under implementationen och för att få en bra översikt så jag kunde hitta fel i min logik i ett tidigt skede.

Backend

Mitt första mål med backenden var att lägga allting i molnet på AWS, insåg snabbt att det inte fanns några stora fördelar med det. Orginalplanen var att ESP kopplar upp sig ett fåtal gånger per dag för att säga till att den fortfarande kan koppla upp sig för att säkerställa

kommunikation och batteri och att den skickar ett meddelande om brevlådan öppnas. Med den mängden meddelanden bedömde jag att det var bättre att köra allting lokalt på en raspberry Pi. Snabbare iterationer av koden och med en sådan liten volym av meddelanden plus att det inte är någonting kritisk gjorde så att jag valde att köra lokalt i största mån.

MQTT. Eftersom jag ville köra det lokalt i docker så tog jag bara der den officiella imagen för en mosquitto server. Det enda som jag behövde lägga till utöver standarden var en konfigurationsfil som nekar all kommunikation som inte följer mTLS med rätt certifikat och att ge containern tillgång till certifikaten den behöver.

Python-script. Den andra containern som behövdes för backend var mitt pythonscript vars mål är att läsa in meddelandena som skickas av ESP över MQTT, avkoda dom och om det behövs skicka en notis över discord i det här fallet. Jag ville också ha en timeout så jag kan skicka en notis att min ESP är död av någon anledning, slut på batteri, skadad elektronik eller bara att den inte lyckas koppla upp sig. Det var ett lätt problem att lösa genom att köra två trådar i pythonscriptet, en som hämtar in och avkodar alla meddelanden och en som bara håller koll på om tillräcklig tid har gått sedan senaste meddelandet. Scriptet kan lätt refaktoreras för att stödja ett nytt API eller flera samtidigt.

Förbättringsområden

ESP32

Projektet hade väldigt kort tidsplan speciellt med jul och nyår mitt i projektet så det var väldigt mycket funktionalitet som jag inte hann med att göra. Några av funktionerna hade bara varit för sakens skull medan några hade gjort det till en mer komplett lösning överlag.

Avståndsmätare. Som tidigare nämnt var lite av hårdvaran valt bara för att jag redan hade delarna och inte för att dom var särskilt bra anpassade för det här projektet. Det största exemplet på det var avståndsmätaren som mest har skapat problem under projektets gång. Den är lite för stor för att kunna monteras på ett bra sätt i brevlådan och den ger inte dom bästa resultaten heller i en såpass instängd miljö. I efterhand hade det nog varit mycket bättre att köpa en lasersnubbeltråd, mycket lättare att tyda den och färre möjligheter för felaktiga data. Mycket av koden nu i ESP'n och i backenden är för att försöka filtrera ut udda mätningar ifrån avståndsmätaren så man inte skickar felaktiga notiser.

FoTA. Fota är något som hade varit kul och praktiskt att få med i projektet. Nu när jag har velat göra kodändringar så måste man gå ut i kylan och flasha om den monterade ESPn.

Säkerhet. Certifikanen ligger inte i en krypterad partition på ESPn, utan bara inbakade i binären, skulle öka säkerheten med att lägga dom i en egen partition som man måste dekryptera för att komma åt dom. Jag har inte heller någon secure boot så den kör gladeligen vilken kod som helst som finns på flashminnet.

Montering. Det var inte det lättaste att montera allting i brevlådan på sättet jag gjorde det med väldigt mycket silvertejp och eltejp. Hade jag haft mer tid på mig hade jag nog försökt montera så mycket som gick i en låda för att underlätta det hela. Här kommer vi tillbaka lite till avståndsmätaren som var det absolut svåraste att montera in då jag var tvungen att fästa den i sidan på brevlådan innan jag kunde koppla in den vilket blev väldigt

pilligt. Det blev att montera avståndsmätaren på sidan istället för locket för att tejpen ville inte fästa där och vinkeln blev lite udda.

Batterier. Min första plan var att köra en gammal powerbank som jag hade liggandes som strömkälla till det hela. Det visade sig väldigt sent då jag inte testade den tidigare att deep sleep inte fungerade när den var inkopplad till powerbanken. Blev tvungen att åka och köpa ett LiPo-batteri i sista minuten för att kunna använda deep sleep utan att den skulle dö helt efter några sekunder av deep sleep. Verkade som att min powerbank trodde ingenting var inkopplat efter några sekunder av det så den ville inte ge ut mer ström.

LoRa. Avståndet nu ifrån brevlådan till min Wi-Fi gateway är på gränsen att den når så den drar ganska mycket mer batteri än vad den skulle behöva. Ibland så misslyckas också uppkopplingen helt så jag tappar meddelanden, vilket är ett problem som skulle behövas lösas också. Just nu försöker den skicka och om den inte lyckas så går den ner i deep sleep igen så meddelandet förloras. Jag har en LoRa sändtagare på min ESP men hade aldrig tid att fixa en LoRa gateway och börja kolla kodmässigt vad som behövs för att få det att fungera.

Device-Shadow. En Device-shadow för konfiguration medan enheten körs hade också varit något som hade varit bra att ha för att kunna ändra på några variabler som sätts upp i konfigurationen innan kompilering.

Backend

Backenden generellt fungerade väldigt bra redan från början. Var väldigt lite kodande som behövdes för hela systemet, det mesta fanns redan färdigt och det var bara en fråga att koppla ihop allting så det fungerade.

Certifikat. Jag vet fortfarande inte varför det händer och hade aldrig tid att undersöka det ordentligt men av någon udda anledning kunde jag inte generera giltiga certifikat på min raspberry pi utan jag var tvungen att göra det på min laptop och kopiera över dom. Det som gör det lite extra udda är att båda körde samma bash script för att generera dom och att båda

har samma version av openSSL. När jag genererade dom på raspberryn så kunde jag inte ens verifiera dom filerna korrekt med openSSLs inbygga funktion för det. Genererade jag dom på min laptop fick jag det förväntade resultatet med att det inte fanns någon CA så fick generera på laptoppen och kopiera över till min raspberry.

Python-script. Här är det enda att det behöver städas upp lite för att underlätta ännu mer att implementera så att jag kan stödja flera API’er. Är bara några fåtal ställen man skulle behöva bryta ut funktionalitet till en funktion för att lätt kunna stödja flera.

Slutsats

I helhet så fungerar projektet okej, med mer tid skulle problemen kunna lösas för en mer robust enhet. LoRa hade nog varit den största förbättring nu när den knappt får någon Wi-Fi täckning inuti brevlådan. Även med alla problem så löser den ändå målet som jag satte mig själv, man får en notis när brevlådan har öppnats så man kan hämta posten, problemet som uppstådde nu när avståndsmätaren dog är ju att man får en notis när man själv hämtar posten. Lösning för att meddela om att enheten har dött ifall den inte kopplar upp sig fungerade utmärkt.

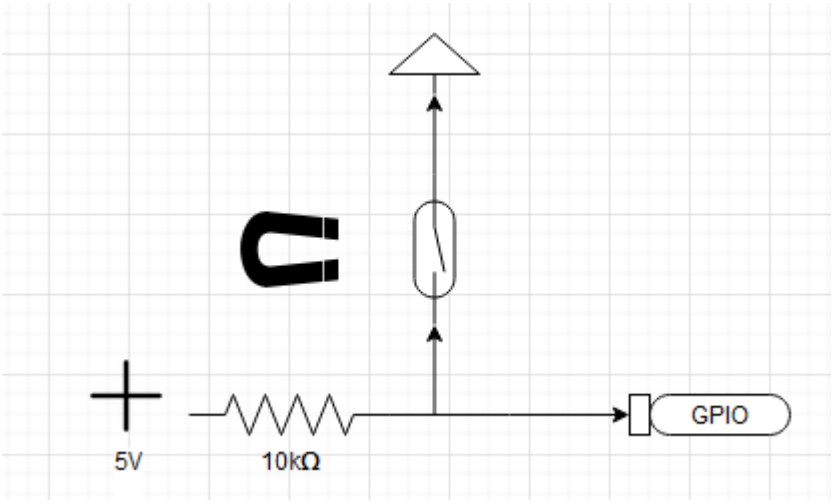
Appendix

Trasig hårdvara

Distanssensorn gick sönder efter 2 dagar utomhus så ändrade programflödet och tog bort koden som användes av den. Hände i princip efter jag hade skrivit klart rapporten så har inte hunnit skriva in mycket mer information om det. Anpassade koden så att den bara håller koll på hur brevlådans lock hanteras.

Figurer

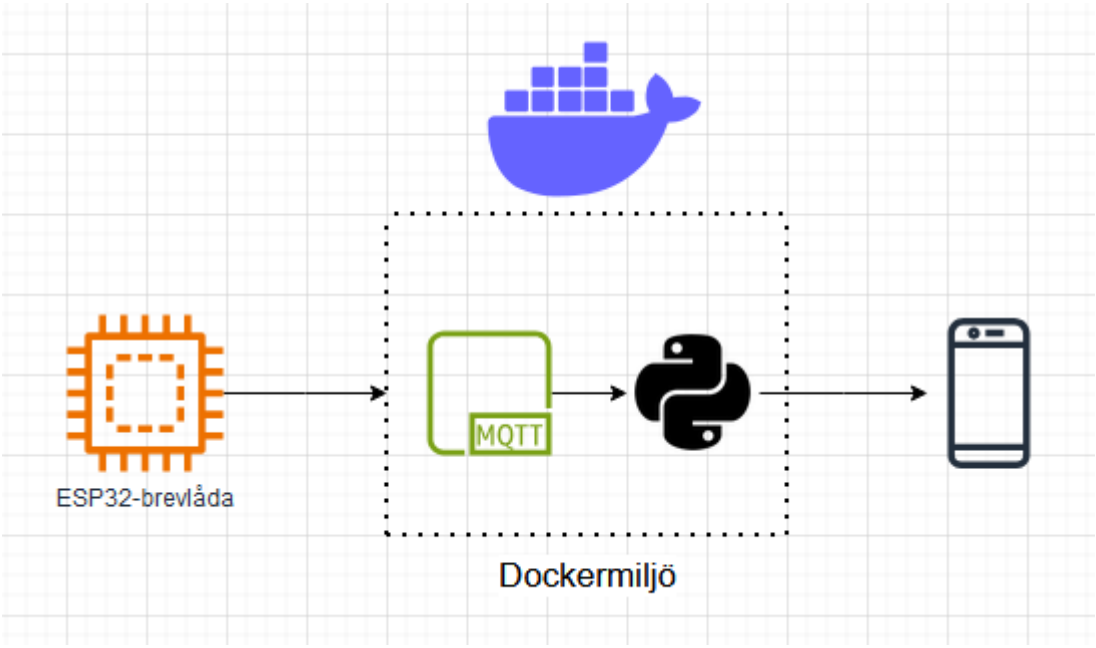
Figur 1. Kopplingsschema för reed-switch.



Anteckningar: Schemat visar bara hur reed-switchen är kopplad till valfri RTC_GPIO port.

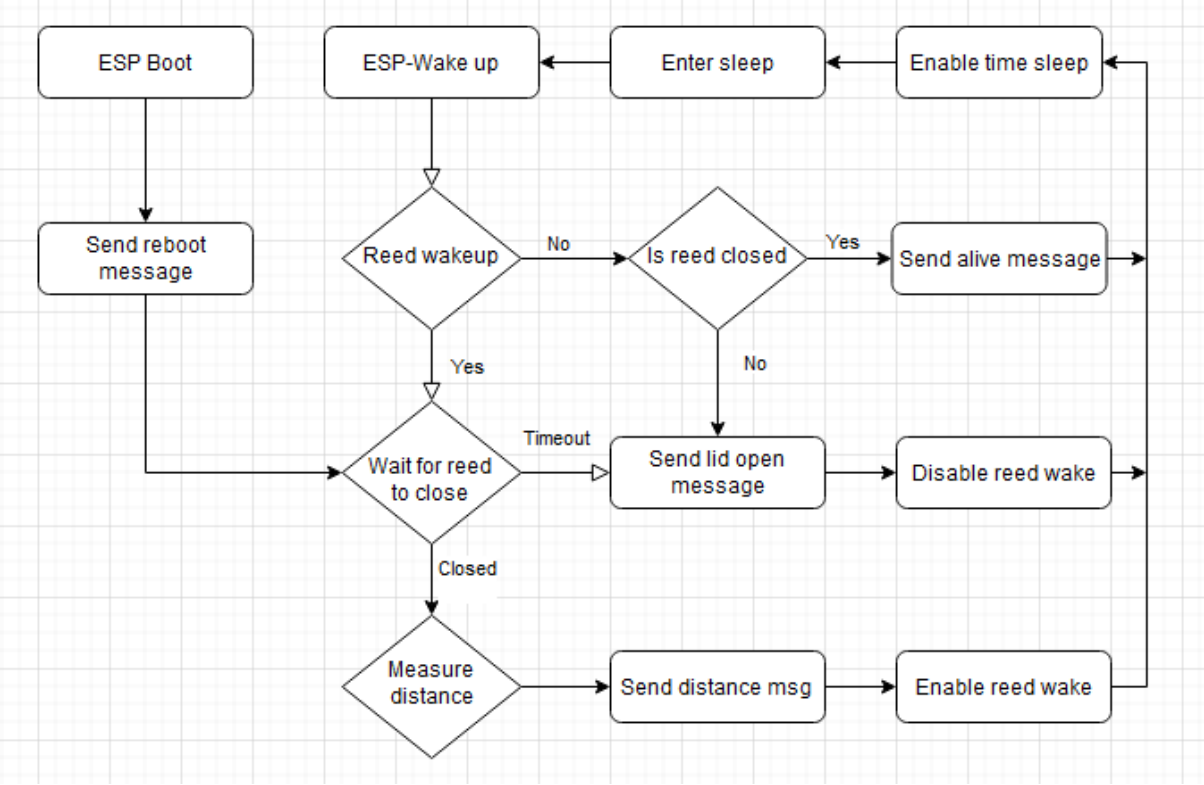
Magneten i figuren är monterad på locket på brevlådan (ej visad i figur).

Figur 2. Projektöversikt.



Anteckningar: Grov översikt av hur projektet är upplaggt.

Figur 3. ESP32 flödesdiagram för logiken.



Anteckningar: Programflödet för ESP koden, icke visat är vad som händer ifall den inte får kontakt med Wi-Fi eller MQTT.