Pager Projekt

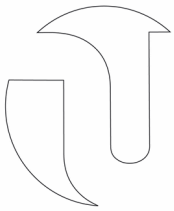
WiFi Modul
Komponentbeskrivning

Abstrakt

Dokumentet beskriver uppbyggnad och struktur över implementationen kommunikationen med WiFi modulen. Modulen har till uppgift att vara en länk mellan hårdvarans processor och en molnbaserad databas som innehåller meddelandedata. Modulen består av ett fåtal huvudfunktioner och flera underliggande hjälpfunktioner.

Versionshistorik

Date	Version	Author	Description
23/05/2020	1.0.0	Elias Johansson	



IVAR JACOBSON CONSULTING

An Essential Unified Process Document

Innehållsförteckning

1 Introduktion

1.1 Dokumentets syfte

Syftet med detta dokument är att ge läsaren en god överblick över systemets struktur och funktionalitet. Dokumentet är avsett för utvecklare som avser att antingen använda modulen i sitt befintliga skick eller för utvecklare som avser att vidareutveckla modulen genom att implementera egna förbättringar eller göra tillägg till modulens grundläggande funktionalitet.

Den information som detta dokument förmedlar hjälper till med följande:

- Att kunna implementera modulen i projektet på enklaste och det mest effektiva sättet möjligt.
- Att kunna underhålla och anpassa modulen för hela systemets behov.
- Lägga till ny och förändra funktionalitet utan att bryta befintlig programexekvering.
- Att enklare kunna testa att modulen och dess funktion är korrekt.

1.2 Avgränsning

Dokumentet begränsas till följande:

- Modulens funktion i systemet och dess huvudsakliga syfte.
- Det aktuellt tillgängliga gränsnittet för modulen.
- Underliggande uppbyggnad och struktur för det aktuella API.
- Konfiguration av modulen i hårdvara.

1.3 Överblick

Dokumentet innehåller följande sektioner:

- **Funktionskrav** – Behandlar det krav som ställs på modulen för att denna skall anses ha fullgod funktionalitet.
- **Implementationsförutsättningar** – De förutsättningar som finns på modulen och dess omgivning som denna skall agera inom.
- **Tillgängliga gränssnitt** – Det kompletta gränssnitt som finns tillgängligt för användaren att använda.
- **Beskrivning WiFi modul** – En beskrivning av den datastruktur och funktionsstruktur som modulen använder sig utav.
- **Testning** – Beskrivning över testmetoden för utvecklingen av modulen till hårdvaran.
- **Förbättringar** – Identifierade samt potentiella brister för framtida bruk som kan vara värt att titta på vid behov.
- **Appendix A – Fullständigt API** – En lista på det aktuella API för modulen som just nu existerar.

2 Funktionskrav

Följande punkter är de krav som ställs på modulen för att den ska anses ha en fullgod funktionalitet:

- Agera som en länk mellan hårdvara och molnbaserad databas.
- Kommunicera med hårdvarans processor genom UART och förmedla kommunikationstatus.
- Kommunicera med molnbaserade databasen genom HTTP begäran för att både hämta och skicka ny information.

För att kunna uppfylla ovan nämnda krav så måste modulen kunna följande:

- Kunna använda det UART-gränssnitt som hårdvaran är uppkopplad på.
- Kunna läsa systeminstruktioner från användaren och översätta dem till korrekt databasinstruktioner.
- Kunna tolka statusmeddelande från både modulen själv samt http begäran och förmedla dessa på ett läsbart sätt till användaren.

3 Implementationsförutsättningar

På grund av den skala på systemet som modulen är anpassad för är kraven på dess miljö väldigt få. Modulen är byggd för en mikrokontroller ifrån STM och använder deras överliggande bibliotek för att abstrahera användaren från att behöva läsa och skriva direkt i register och därmed också öka portabiliteten.

3.1 Framework

Modulen bör fungera med samtliga mikrokontroller från alla STM32 familjer under förutsättning att HAL, Hardware Abstraction Layer, samt CMSIS används för att undvika att använda specifika registeradresser och namn från en särskild familj eller typ. Det som då behövs anges är:

- Konfigurationen av pins för det UART gränssnitt man tänker använda.
- Ange vilket UART-gränsnitt man valt att använda.
- Konfigurera rätt dataöverföringshastighet, dvs Baudrate till 9600.
- Övriga parametrar ska låtas vara grundinställningar.

3.2 Språk

Modulen är programmerad i C och är det som stödjs bäst för denna typ av enhet. Till viss del kan visst C++ stöd finnas men man måste då ta hänsyn till att man kan skada portabiliteten och att man måste noggrant säkerställa vad som har stöd och vad som inte kommer fungera.

3.3 Utvecklingsdesign

Samtliga funktioner har små specifika uppgifter som ska utföras med undantag av vissa samlingsfunktioner som använder flera av dessa för att utföra en större uppgift. Alla funktioner ska beskrivas tydligt i header filen vad dess syfte är, vilka argument som ska skickas med och vad som funktionen returnerar. Om tillämpligt skall funktionen returnera status över huruvida kommunikationen var lyckad eller om

möjligt vad som har gått fel. Detta för att möjliggöra goda friheter att implementera felhantering i exekveringsflödet för att undvika problem där systemet kan låsas.

Designen för modulen är uppbyggd på att att ge användaren ett lättanvänt API. Dvs att även om modulen innehåller många mindre funktioner som har ett specifikt syfte och ändamål så vill man presentera ett gränsnitt för användaren som är så litet och enkelt som möjlighet men ändå tillhandahåller all nödvändig funktionalitet. Med detta i åtanke så finns det två typer av funktioner. Så kallade wrapper-funktioner som har ett större syfte och utför mer arbete. Den andra typen är hjälpfunktioner vilket är små väldigt specifika funktioner som wrapper funktioner använder sig utav.

4 Tillgängliga gränsnitt

Modulen tillhandahåller gränsnitt för att kommunicera med molnbaserade databasen på rätt sätt. Användaren ska kunna utnyttja modulen utan att veta exakt hur den kommunicerar med databasen och exakt vilka parametrar denne kräver. Användaren ska enkelt kunna skicka med endast relevant information som ska hämtas eller skickas och modulen ska därmed se till att detta hamnar på rätt plats utan ytterligare information från användare. Därför består modulen av flera olika funktioner som har till uppgift att helt enkelt bygga upp kommandon eller som läser av och fördelar upp det svar man får på ett betydligt mer hanterbart sätt. Samtliga funktioner hittas i bilaga A.

5 Beskrivning – WiFi modul

För att tillhandahålla kraven på att lista resultat från användningen av gränssnittet på ett läsbart sätt samt för att kunna göra smidig felhantering krävs det att underliggande struktur är genomtänkt för ändamålet.

5.1 Datastrukturer

För enklare felhantering så returnerar de funktioner som tillämpar någon form av kommunikation med Wifi ett felmeddelande av en särskild typ. Denna består i sin enkelhet av en enum struktur enligt nedan:

```
typedef enum {
        wifi_ok,
        wifi_error,
        wifi_timeout,
        wifi_noAP
} wifi_status;
```

Med hjälp av en variabel av denna struktur kan man enkelt kontrollera vilken typ av status av interaktionen som funktionen returnerar. wifi_ok är den vanligaste och den man önskar se då det betyder att allt har gått bra. wifi_error innebär att http begäran gick fel och att modulen har returnerat texten error till hårdvaran. wifi_timeout i sin tur innebär att det inte nödvändigtvis blev ett direkt fel men att max tid för väntan på svar gick ut före att ett svar kom. Orsaken för detta får man dock undersöka på egen hand. Slutligen har vi en specifik typ av kod som är wifi_noAP. Denna används

endast vid ett tillfälle vilket är då man använder funktionen för att lista tillgängliga accesspunkter och det inte finns några eller då inga kunde hittas.

En särskild struktur används för att spara meddelanden som hämtats från databasen vilken i sin tur omfamnas av ytterligare en struktur för att spara metadatat för meddelandelistan. Varje meddelande sparas individuellt i en strukt enligt nedan:

```
typedef struct message {
        uint8_t title[40];
        uint8_t msg[max_msg_size];
        uint8_t sender[50];
        uint8_t read[10];
        uint8_t cat[8];
} message;
```

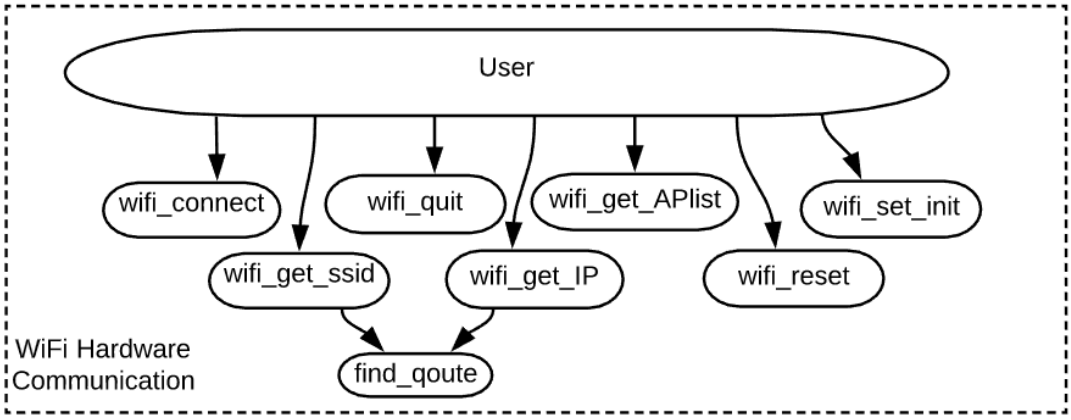
Denna innehåller därmed utöver meddelandet i sig titeln, avsändare, den kategori meddelandet tillhör (Läst/Oläst) samt huruvuda det faktiskt är läst eller ej ännu.

Dessa meddelande lagras sedan i en omfamnande struktur som består av en lista av ovan strukt tillsammans med en variabel som helt enkelt håller räkningen på antalet meddelanden enligt nedan:

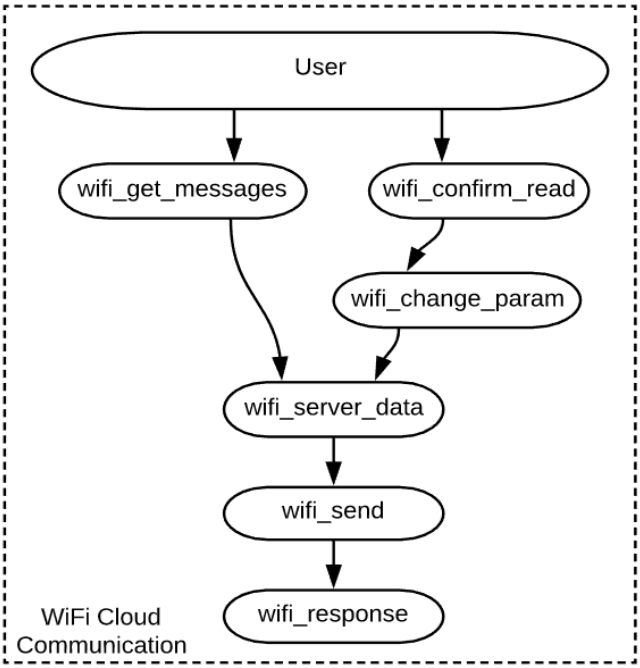
```
typedef struct messages_collection {
        message messages[max_msgs];
        uint8_t count;
} messages_collection;
```

5.2 Uppbyggnad funktioner

Man skulle kunna dela upp samtliga funktioner i två delar. Vi har dels de delar som kommunicerar enbart med wifi modulen för att till exempel göra inställningar eller för att läsa ut status angående wifi anslutningen. Vi har också de typer av funktioner som kommunicerar med wifi modulen genom att skicka instruktioner om uppkoppling till databas för att hämta eller skriva information. I det förstnämnda har vi bara ett lager av funktioner (med ett litet undantag, find_qoute), dvs att användaren kallar dessa individuellt för att hämta någon särskild information. Överblick av dessa syns nedan:



För kommunikationen till databasen har vi dock flera lager av funktioner som används olika beroende på vilket jobb man vill utföra. Det innebär att användaren använder endast “wrapper” funktionerna som i sin tur sedan tar hand om att generera korrekt kommando att skicka till wifi modulen och som sedan också tar hand svarssträngen och delar upp den så att resultatet blir lättare att jobba med. De olika lagrena syns tydligt i diagrammet nedan:



6 Testning

Testning utav utvecklingen av modulen föll ner i två olika delar. Första delen var att upprätta kontakt mellan hårdvaran och wifi modulen så att de har fullgod kommunikation med varandra. Därefter var nästa steg att upprätta kontakten mellan wifi modulen och databasen. En TDD, Test Driven Development, utvecklingsmetod användes eftersom det var svårt att göra automatiserade tester när det finns så många olika lager som kan fela. Det handlar inte bara om att testa sin kod utan även kommunikationen i sig måste fungera, båda fysiskt via kablarna samt mjuvarumässigt följa protokollen. För att underlätta testningen så utvecklades först ett CLI, Command

Line Interface, så att man manuellt kan skriva kommandon till wifi modulen och därmed läsa upp det svar man fick. Det hjälpte enormt mycket då det på förhand inte var känt vilka svar vi skulle få oavsett om interaktionen var lyckad eller misslyckad. Först när wifi modulens kommunikationsprotokoll var fastställt kunde testfunktioner upprättas.

Då TDD metoden väl var på plats så kunde funktionsanrop skapas som hade givna argument och där målet var att ett särskilt svar skulle returneras. På detta viset kunde man sedan förfina koden och konfigurera anropen tills dess att funktionen gav fullgoda svar. Den stora utmaningen här var att hantera den enorma uppsjö av problem som skulle kunna uppstå. Det kan bli flera olika typer av fel i flera olika länkar så som till exempel i UART mellan hårdvara och modul, i wifi uppkopplingen för modulen, i uppkopplingen till databas eller i kommunikationen mellan modul och databas. Övriga fel så som korrupt data eller liknande är också möjligt om än risken är minimal. I dessa fall var det viktigt att hitta en bra kompromiss över vad som är rimligt att behöva hantera för att uppnå fullgod funktionalitet för det syfte som produkten ska uppfylla.

7 Förbättringar

Flera förbättring kan givetvis göras och flertalet av dem är direkt kopplat till den tidigare nämnda felhanteringen. Funktionerna gör det som de ska göra med fullgott resultat så länge samtlig kommunikation fungerar korrekt. Man kan identifiera flera scenario där vissa specifika fel skulle bryta programexekveringen och i absolut värsta fall skulle mjukvaran hänga upp sig och en hård omstart hade varit oundvikligt.

Utöver en mer komplett felhantering så kan man absolut även utöka funktionaliteten men den hänger ihop med produktens syfte och den nivå som kraven på denna ställer.

Bilaga A – Fullständigt API

```
* @brief Connect to wifi
* @param uint8_t* ssid The name of the wifi AP
* @param uint8_t* pass The password of the wifi AP
* @return wifi_status Status of the wifi interaction
wifi_status wifi_connect(uint8_t* ssid, uint8_t* pass);

* @brief Quit current AP connection
* @return wifi_status Status of the wifi interaction
wifi_status wifi_quit();

* @brief reset wifi module
* @return wifi_status Status of the wifi interaction
wifi_status wifi_reset();

* @brief Get SSID of connected wifi
* @param uint8_t* res The c string to save the name to
* @return wifi_status Status of the wifi interaction
wifi_status wifi_get_ssid(uint8_t* res);

* @brief Get IP address of connected wifi
* @param uint8_t* res The c string to save the IP address to
* @return wifi_status Status of the wifi interaction
wifi_status wifi_get_IP(uint8_t* res);

* @brief Get list of available AP
* @param list Pointer to the string array to save the AP SSIDs
* @return cnt Number of APs found
uint8_t wifi_get_APlist(uint8_t (*list)[30]);

* @brief Set init values (Single connection, Station mode)
* @return wifi_status Status of the wifi interaction
wifi_status wifi_set_init();

* @brief Receive data by using wifi_send_data
* @param uint8_t* data Where to store the data
* @param uint8_t* cat The category from which to get messages
* @return wifi_status Status of the wifi interaction
wifi_status wifi_get_messages(messages_collection* data, uint8_t* cat);
```

```
* @brief Confirm read status on an unread message
* @param message* The message to be confirmed
* @return wifi_status Status of the wifi interaction
wifi_status wifi_confirm_read(message* msg);

* @brief Change data by using wifi_send_data
* @param uint8_t* data data to be sent
* @param uint8_t* cat The category to send the data to
* @param uint8_t* message The message to change
* @param uint8_t* param The parameter to change
* @return wifi_status Status of the wifi interaction
wifi_status wifi_change_param(uint8_t* cat, uint8_t* message, uint8_t* param, uint8_t* data);

* @brief Send and receive data from cloud
* @param uint8_t mode Set send or receive mode
* @param uint8_t* json URL to json file to read or write
* @param uint8_t* res String pointer with data to send or where to save received
* @return wifi_status Status of the wifi interaction
wifi_status wifi_server_data(uint8_t mode, uint8_t* json, uint8_t* res);

* @brief Get the wifi response after sending AT command
* @param uint8_t* res The c string to save the response to
* @return wifi_status Status of the wifi interaction
wifi_status wifi_response(uint8_t* res);

* @brief Send uart transmission to wifi module
* @param command Command to be sent
* @param res Pointer to response string
wifi_status wifi_send(uint8_t* command, uint8_t* res);

* @brief Find text within " " starting from search parameter
* @param uint8_t* str Message to search within
* @param uint8_t* begin String to start from
* @param uint8_t end Character to end the quote
* @param uint8_t* res Where to save the resulting string
void find_quote(uint8_t* str, uint8_t* begin, uint8_t end, uint8_t* res);
```