NACKADEMIN

Projectet – Hantera en LED med UART-protokollet

Inbyggda system: arkitektur och design

Kursansvarig: Ludwig Simonsson

Rana Alwan - Rana.Alwan@yh.nackademin.se

Innehåll

Introduktion

Denna rapport beskriver utvecklingen av en drivrutin för UART-kommunikation på STM32F411x-plattformen. Syftet med rapporten är att ge en detaljerad beskrivning av designprocessen för att utveckla och implementera en högkvalitativ drivrutin som kan hantera kommunikationen mellan mikrokontrollern och andra enheter som använder UART-protokollet.

UART är ett av de mest använda seriella kommunikationsprotokollen inom elektronik och IoT, och det har en rad fördelar, såsom att det är enkelt och kräver mindre minne och hårdvara än andra protokoll, som USB och WiFi. UART består av en sändare och en mottagare som överför data bit för bit över en kommunikationslinje.

Målet med denna rapport är att erbjuda en grundlig beskrivning av utvecklingsprocessen för en högkvalitativ drivrutin för UART-kommunikation på STM32F411x-plattformen. Vi kommer att beskriva hur drivrutinen har utformats och implementerats på plattformen för att säkerställa att den är pålitlig och effektiv i hanteringen av kommunikationen.

Rapporten kommer att innehålla en detaljerad beskrivning av utvecklingsprocessen, inklusive beskrivningar av de olika delarna av drivrutinen och hur de fungerar tillsammans för att uppnå målet. Vi kommer också att beskriva hur drivrutinen har testats och utvärderats för att säkerställa att den möter kraven på tillförlitlighet och effektivitet.

Genom att läsa denna rapport kan läsaren få en grundlig förståelse av hur man utvecklar en högkvalitativ drivrutin för UART-kommunikation på STM32F411x-plattformen, samt hur man kan använda drivrutinen i praktiska tillämpningar.

Dagbok

Dag 1:

Jag började dagen med att läsa igenom projektuppgiften noggrant och skapa en plan för hur jag skulle gå tillväga. Efter att ha identifierat de olika delarna i uppgiften satte jag upp en tidsplan för när jag skulle utföra varje del. Jag började sedan med att göra en översiktlig plan för hur jag skulle designa systemet och vilka komponenter jag skulle använda.

Idag var första dagen på projektet och fokus låg på att bekanta sig med dokumentationen och förstå plattformens egenskaper och begränsningar för att utveckla en pålitlig och effektiv drivrutin för UART-kommunikation på STM32F411x-plattformen.

Jag började dagen med att studera den tekniska dokumentationen för STM32F411x-plattformen och UART-protokollet för att få en bättre förståelse för dess funktionalitet. Jag lade särskilt fokus på att förstå hur UART-protokollet fungerar och hur det kan integreras med STM32F411x-plattformen.

Under resten av dagen läste jag igenom databladen för de olika komponenterna på STM32F411x-plattformen, inklusive UART-modulen. Jag studerade varje register och dess funktioner för att få en klar bild av hur jag skulle kunna använda dem i min drivrutin.

Under arbetet stötte jag på vissa utmaningar och frågor som jag behövde ta itu med. Till exempel fann jag det svårt att förstå vissa funktioner i UART-modulen på grund av oklarheter i dokumentationen. För att lösa detta problem behövde jag söka hjälp från mina klasskamrater och andra tekniska källor för att få en bättre förståelse.

Trots dessa utmaningar var det en produktiv dag och jag känner mig mer förberedd och säker på vad som krävs för att utveckla en framgångsrik drivrutin för UART-kommunikation på STM32F411x-plattformen. Jag ser fram emot att fortsätta mitt arbete med projektet och ta itu med nästa steg i utvecklingsprocesse..

Dag 2:

På dag två fortsatte jag med att designa systemet i mer detalj. Jag identifierade vilka enheter som skulle användas och skapade en skiss över hur de skulle kopplas samman. Jag började också skriva kod för att testa säkerställa att de fungerade korrekt.

utvecklingsdagboken handlade om att skriva själva drivrutinen för UART-kommunikation på STM32F411x-plattformen. Arbetet började med att använda den samlade filen "UART.h"

som innehåller alla nödvändiga headers och funktioner för att implementera UART-protokollet.

För att initiera USART-protokollet och dess komponenter skrevs först en funktion som heter "USART2_Init". I denna funktion aktiverades klocktillgången för UART2 genom att sätta bit 17 i APB1ENR till 1 och GPIOA genom att sätta bit 0 i AHB1ENR till 1. Alternativ funktion för valda pinsen PA2 och PA3 valdes också, tillsammans med typen av alternativ funktion för de valda pinsen.

Efter att enhetens kommunikation var konstruerad konfigurerades UART:en genom att sätta standard baud-rate till 9600bps med hjälp av hexadecimalen 0x0683. Tx och Rx konfigurerades också för att arbeta i 8 bitars data, 1 stopp-bit och ingen paritet, genom att sätta bit 3 och bit 2 i CR1 till 1. Därefter nollställdes CR2 och CR3 och bit 13 (UART-aktiveringen) ställdes om till 1 för att aktivera UART:en.

För att skriva funktionerna "USART2_write" och "USART2_read" skrevs först en skrivfunktion som överför data till terminalen och sätter kravet att statusen på överföringen måste vara tom och redo att ta emot nästa byte innan en ny byte kan skickas. Bit 7 i SR användes för att kontrollera statusen och överföringen av byten till dataregistret gjordes genom att sätta ch & 0xFF i DR. Läsfunktionen "USART2_read" läser in data som skickas till mikrokontrollern. Bit 5 i SR användes för att kontrollera om det finns mer data att hämta och datan returneras sedan från DR.

Totalt sett tog det ungefär en dag att utveckla själva drivrutinen för UART-kommunikation på STM32F411x-plattformen. Detta var en viktig del av projektet och resultatet var tillfredsställande.

Dag 3:

Dag tre fokuserade jag på att integrera enheterna och skapa en grundläggande funktionalitet för systemet. Jag skrev kod för att läsa in data och skicka den till en central enhet för bearbetning. Jag fortsatte också med att testa systemet och felsöka eventuella problem som uppstod.

Det låter som en produktiv dag där jag har fokuserat på att utveckla en periferi-drivrutin som fungerar som ett komplement till UART-drivrutinen. Att använda en klass för detta ändamål är en bra strukturerad metod för att hålla koden organiserad och lätt att underhålla.

Jag definiera makron för att ange pins och mode-bitar är ett bra sätt att göra koden läsbar och lättförståelig. Detta hjälper också till att göra koden mer portabel, eftersom dessa inställningar kan variera mellan olika enheter.

Jag har använd en enhetsspecifik headerfil för att ange hårdvaruspecifika inställningar är också en bra praxis. Detta hjälper till att abstrahera bort de specifika hårdvarudetaljerna från resten av koden och gör koden mer lättförståelig.

Jag implementera en konstruktor för Led-klassen som tar färg och tillstånd som parametrar är också en bra praxis eftersom det ger flexibilitet för att skapa objekt med olika egenskaper. Det är också bra att ha funktioner för att ställa in och hämta tillståndet för LED-lampan.

Dag 4:

På dag fyra strukturerade jag dokumentationen och skapade ett Github-repo för att organisera och dela information om projektet. Jag skapade en README-fil som innehöll en kort beskrivning av projektet och en lista över de komponenter och enheter som användes. Jag skapade också en struktur för dokumentationen och lade till relevanta dokument i separata mappar i repo:t. För att göra dokumentationen mer lättillgänglig och lättförståelig lade jag till en beskrivning av varje dokument och dess syfte i README-filen, samt länkar till de relevanta filerna.

Det var helt dedikerad till att strukturera dokumentationen och skapa ett Github-repository. Först skapade du ett Github-konto om du inte redan hade ett och sedan skapade du ett nytt repository för ditt projekt. Det är viktigt att namnge ditt repository på ett sätt som gör det lätt att identifiera vad projektet handlar om och inkludera en kort beskrivning för att göra det mer begripligt för andra användare.

Efter att ha skapat repository:t laddade du upp all befintlig kod som du hade skrivit hittills till repository:t. På så sätt är all kod för projektet samlad på en enda plats och det är enkelt för andra att se vad som har gjorts hittills.

För att göra det lättare att organisera och dela information om projektet var det viktigt att strukturera dokumentationen. Du definierade vilka typer av dokument som skulle inkluderas och skapade en struktur för dem. För varje område skapade du en separat mapp i repository:t och lade till relevanta dokument.

För att göra dokumentationen mer lättillgänglig och lättförståelig, la du till en beskrivning av varje dokument och dess syfte i README-filen. Du inkluderade också länkar till de relevanta filerna för att göra det enkelt för andra att hitta informationen de behöver.

Efter att ha strukturerat dokumentationen och laddat upp all befintlig kod på Github var det enkelt för utbildaren att få tillgång till all information som de behövde för att bedöma projektet.

Resultat

För kodfiler med kommentarer och förklaringar, se Github-repo:

https://github.com/ranaalwan/Project_LED_med_UART_protokollet

UART.h

Denna fil utgör en header-fil för UART där relevanta bibliotek inkluderas och två funktioner definieras.

UART.c

Denna fil utgör UART-protokollet. Först inkluderades relevanta bibliotek. Sedan initierades USARTfunktionen
som i sin tur initierade USART-protokollet. USART användes för att specificera en funktion som kan köra både synkront och asynkront. Därefter följde fyra steg.

1.Enabla klocktillgång för UART2. Genom att titta i blockdiagrammet i dokumentationen så såg
att USART2 hade APB1 som buss. I referensmanualen, under klockan, så förtydligades att det, var RCC som skulle användas och klockan ska sättas till 1 för att vara på. UART2 aktiverades sedan genommatt sätta bit 17 till 1. Med hjälp av hexadecimaler.

2. Enabla klocktillgång för port A. Ungefär på samma tillvägagångssätt enligt punkt 1 aktiverades GPIO genom att sätta bit 0 till 1.

3. Enabla pins relaterade till porten genom att i referensmanualen kolla på mode register. Detta utfördes genom att först rensa bitarna 4-7 innan vi aktiverade dom. Rensning utfördes för att säkerställa utifall dom var i andra lägen.

4. Val av alternativ funktion för de valda pinsen. Fanns ”low” och ”high” som variation för hur bitarna ska definieras men i aktuellt fall sattes en nolla, AFR[0], för att visa att alternativ funktion skulle användas och börja från början. Sedan gjordes en konfiguration av UART där vi kontrollerade och satte en baud rate på 9600 bps, gav
kontrollregister 1 aktiverad sändare och mottagare att arbeta med 8-bitarsdata samt att register 2 och 3 är nollställda. Sedan kunde UART aktiveras igen.
Ovanstående har resulterat i att UART kan användas.
Sedan skulle write- och read-funktioner göras vilket inleddes med att skapa write-regler. Till denna deklarerades write-funktion som kontrollerade att statusen på dataöverföringsregistret (bit 7) var tom och kan ta emot nästa byte och då sätter överföringen av byten till registret.
Därefter gjordes på liknande sätt en read-funktion som med hjälp av en while-loop istället kontrollerar ifall det finns mer data att hämta och då läser ut datan.
Därefter strukturerades överföringsströmmarna med hjälp av en struct.
Sedan gjordes två funktioner. Den första funktionen läste från konsolen och den andra skrev ut till konsolen och slutligen gjordes en funktion för att testa läs- och skrivfunktionerna.

LED.h

Denna fil utgör en header-fil till LED. Här inkluderades relevanta bibliotek samt headern från UART.
Vidare sattes ett antal definitioner vilka i huvuddrag var:
• Vilken GPIO som ska vara ansvarig för LED-funktionen.
• Klocksignalen för porten.
• Pins och mode bits för respektive LED-färg.
• Två enums för färg och status.
• En struct bestående av variablerna för färg och status.
• Funktioner för LED-konstruktorn och kontroll av status.

LED.c

Denna fil utgör C-filen där funktionerna, inkluderade från LED-headerfilen skrivs.
I den första funktionen, konstruktorn för LED-lamporna, skapades först två konstanter för färg och
status som pekar på adressen till headern. Sedan enablas klockan. Därefter skrevs konfiguration av
LED-pinsen beroende på färg med hjälp av ett switch-statement. Reglerna utfördes så att om färgen
exempelvis var röd och om statusen är satt till ”på” så sattes LEDen på, annars till av. Detta case
gjorde för alla färger.
Nästa funktion, som ska kontrollera statusen, var uppbyggd med liknande konstruktion som den
första funktionen och tillämpades på alla färger, med hjälp av ett switch-statement kontrollerades att färgen exempelvis var röd och om statusen är så definierades pinsens output till
aktiv, annars inaktiv.
Avslutningsvis i denna fil skapades en funktion för att kontrollera färgen på en LED och printa vilken
färg LEDen är satt till.

Main.c

Här testkörs all kod genom att inkludera våra filer och sedan skapades två LED-lampor. Sedan sattes
färgen och statusen på dessa.

Github

I Github skapades ett repository med struktur enligt anvisningarna.

https://github.com/ranaalwan/Project_LED_med_UART_protokollet

innehållandes undermappar för:

• Hårdvarumapp – Dokument avseende hårdvaran som projektet konstruerats för.
• Mapp för källkod – All kod.
• Mapp med rapport – rapport.

Källhänvisning

-STM32F411xC STM32F411xE – STMicroelectronics

-UART Communication Protocol - How it works? - Codrey Electronics

-RM0383 Reference Manual– STMicroelectronics

-UM1724 User manual – STMicroelectronics

- Rapidtables.com/convert/number