NACKADEMIN

2023-05-14
IOT22

Inbyggda System: Arkitektur och Design

- *Hantera en LED med UART-protokollet: Drivrutins-konstruktion och programmering i C/C++*

Namn: Martin Myrberg
E-post: Martin.Myrberg@yh.nackademin.se
Länk till projekt: https://github.com/mmyrberg/LED-UART

Innehållsförteckning

1. Inledning

UART (Universal Asynchronous Receiver/Transmitter) är ett kommunikationsprotokoll som används för att överföra data seriellt (bit för bit) mellan elektroniska enheter. Det används ofta inom inbyggda system för att upprätta kommunikation mellan exempelvis en mikrokontroller och en dator eller i sensorer och aktuatorer för att överföra data till en mikrokontroller som i sin tur tolkar och hanterar datan. I förhållande till andra kommunikationsprotokoll, som I2C och SPI, är UART vanligtvis långsammare men stödjer kommunikation på längre avstånd mellan enheter och driver samtidigt mindre minne och hårdvara. Som en följd av detta är UART ett populärt val för grundläggande seriell kommunikation.

UART består av en sändare (transmitter) och en mottagare (receiver). Mellan dessa finns en baudrate-generator som kontrollerar överföringshastigheten. Den asynkrona egenskapen innebär att det inte finns någon gemensam klocka mellan sändaren och mottagaren. Istället används en startbit för att indikera början av en överföring, följt av en eller flera databitar, en parity samt en stoppbit som avslutar överföringen. Det finns dock en variant av UART-protokollet som kallas USART (Universal Synchronous/Asynchronous Receiver-Transmitter) som även har stöd för synkron kommunikation. Här finns en gemensam klocka mellan sändare och mottagare för att synkronisera överföringen och data kan därmed överföras i block vilket gör kommunikationen snabbare.

I detta projekt utvecklas en drivrutin med UART-protokollet för att kommunicera mellan en mikrokontroller (stm32F411x) och en LED-lampa. I mjukvaran, skriven i programspråket C, aktiveras dock klocktillgång för UART vilket innebär att USART-protokollet initieras för att möjliggöra både synkron och asynkron kommunikation. Denna rapport beskriver arbetsprocessen för att implementera och utforma en sådan drivrutin på STM-plattformen, samt en närmare inblick i den programvara som studerats, i syfte att fördjupa kunskaper inom arkitektur och design för inbyggda system.

2. Arbetsprocess/Genomförande

Projektet inleddes med att bekanta sig med STM-plattformens dokumentation. I detta skede handlade det mycket om att rent generellt försöka läsa på och lära sig att hitta i dokumentationen vilket var en utmaning i sig. När viss vana ändå infunnit sig, gick jag vidare och fokuserade på att läsa på om de delar som specifikt handlade om UART-protokollet och hur det kunde integreras med STM-plattformen. Den största utmaningen jag stötte på i samband med detta var att lyckas komma underfund med hur jag skulle gå från dokumentationens beskrivning av olika register till att översätta det i kod och vilka bitar som behövde manipuleras för att uppnå önskad funktion. Under tiden pågick även föreläsningar och laborationer i kursen, som visserligen underlättade förståelsen, men jag behövde ändå bolla med mina studentkollegor och söka information i andra källor för att lyckas komma vidare och påbörja arbetet med själva drivrutinen.

Innan kodarbetet med drivrutinen kom igång, startade jag upp ett nytt Github repository och länkade till min lokala filstruktur (se kapitel 3.2) i syfte att kunna dela och redovisa projektet på ett välstrukturerat sätt. Detta repository uppdaterades sedan kontinuerligt under arbetets gång och gav

bra övning/repetition i versionshantering för projekt inom inbyggda system men även för mjukvarurelaterade projekt i största allmänhet. Innan projektinlämning, lades även en README-fil till för en övergripande beskrivning av projektet där jag också passade på att träna mina färdigheter i Markdown-formatet.

När arbetsprocessen sedan övergick till inriktning på drivrutinsutvecklingen, bestod första steget av att utveckla UART-drivrutinen. Koden för detta organiserades upp i två filer: UART.h och UART.c (se kapitel 3.1.1 och 3.1.2). I headerfilen (UART.h) definierades alla funktioner, variabler och andra deklarationer som behövdes för att implementera UART-protokollet. I UART.c implementerades sedan funktionerna som hade definierats i headerfilen. Huvudfunktionen i denna fil, ”USART2_init”, initierade USART-protokollet som variant på UART för att kunna köra både synkron och asynkron kommunikation.

I nästa skede av arbetsprocessen gjordes ytterligare en drivrutin med uppgift att kunna styra LED-lampan bestående av två filer: LED.h och LED.c (se kapitel 3.1.3 och 3.1.4). Dessa innehöll instruktioner för att styra LED-lampant tillstånd, av- eller påslagen, samt logik för att konfiguera LED-pinsen beroende på önskad färg på LED-lampans ljus (rött, gult, blått, grönt). Tillsammans med UART-drivrutinen började därmed en komplett programvara ta sin form för att kunna interagera med STM-plattformen och uppnå önskad funktion.

En av de viktigare delarna i arbetets gång var att kommentera ut koden och beskriva varje rad med förklarande kommentarer. Här redogjordes inte bara för **vad** varje rad gjorde men också **varför** koden såg ut som den gjorde. Denna dokumenterande process var en stor del i projektets lärande och ett betydelsefullt moment för att uppnå efterfrågade kriterier. Syftet var att en utomstående part snabbt skulle kunna kliva in i vår programmeringsmiljö och anta det arbete som utförts. Det krävdes dock en hel del samarbete tillsammans med mina studentkollegor i grupp för att förstå alla delar och lyckas kommentera ut varenda rad. Dessa möten var till stor hjälp för att klargöra frågetecken och underlättade att komma vidare i arbetet.

Avslutningsvis lades en enhetsspecifik headerfil till (stm32f4xx.h) för att ange särskilda inställningar för hårdvaran samt en main-fil för att initiera programstart och anropa relevanta funktioner som konstruerade efterfrågade attribut på LED-lampan. Därefter strukturerandes alla filer i mappstrukturen och slutligen skrevs denna rapport för att sammanställa och redovisa projektets olika delar.

3. Resultat

Detta avsnitt beskriver koden i detalj för den färdiga produkten samt användning, hantering och struktur av tillhörande Github-repo. För hela programvarans kod med tillhörande kommentarer och förklaringar, se: https://github.com/mmyrberg/LED-UART

3.1 Programstruktur

Programmet består av totalt fem filer enligt nedan som tillsammans utgör ett färdigt program för att hantera en LED med UART-protokollet. För att underlätta användning, felsökning och testning

(simulering) rekommenderas STM32CubeIDE vilket är en integrerad utvecklingsmiljö (IDE) från STMicroelectronics som använder STM32-arkitekturen.

3.1.1 UART.h

Headerfil som knyter samman UART.c och Main.c och innehåller bland annat deklarationer av funktionerna ”USART2_init” och ”test_setup”. Observera den senare inte används för LED-funktionen utan är en testfunktion för att kontrollera att UART-kommunikationen är rätt konfigurerad och fungerar som den ska. Vidare inkluderas biblioteken för den enhetsstpecifika headerfilen stm32f4xx.h och standard I/O (stdio) i C.

3.1.2 UART.c

I denna fil implementerades och definierades funktionerna som utgjorde UART-drivrutinen. Den första och kanske mest centrala funktionen här var ”USART2_init”-funktionen. Denna funktion hade till uppgift att initiera USART-protokollet och dess beståndsdelar. Funktionen byggdes upp i fyra följande steg:

1. **Aktivera klocktillgång för USART2.**
2. **Aktivera klocktillgång för GPIO-port A.**

I steg ett och två aktiverades klocktillgången för USART2 och GPIO-port A genom att ställa om bitarna på respektive aktuellt register till 1 med hjälp av bitoperatorer och hexadecimaler (något som användes genomgående i koden för att skapa drivrutinen). Med hjälp av referensmanualen gick det att se att APB1ENR och AHB1ENR var de register inom RCC (systemklockan) som skulle användas för att aktivera (eller inaktivera) klockan.

3. **Konfiguera pinsen PA2 och PA3 som ut- respektive ingång.** Detta gjordes genom att ändra inställningarna för bitarna i mode-registret genom att först rensa bitarna 4–7, ifall dessa var i andra lägen, och sedan sätta bitarna 5 och 7 till 1 för att aktivera pinsen.
4. **Konfiguera alternativ funktion för PA2 och PA3** genom att först rensa bitarna 8–15 för att förbereda pinsen och sedan sätta bitarna 8–11 och 12–14 till formatet 0111. Detta gav GPIO-pinsen möjlighet att användas för fler syften, än bara enkel in-utdata, exempelvis kommunikation med UART-protokollet.

När enhetens kommunikation hade konstruerats enligt ovan konfigurerades UART med en baudrate på 9600 bitar per sekund (bps) genom att skriva hexavärdet 0x0683 till BRR-registret. Sedan sattes bit 2 och 3 i kontrollregister 1 (CR1) till 1 för att aktivera sändare (tx) och mottagare (rx) att arbeta i 8-bitars data, med 1 stoppbit och ingen paritet. Därefter nollställdes kontrollregisterna 2 (CR2) och 3 (CR3) och slutligen sattes bit 13 i CR1 till 1 för att aktivera USART2-modulen.

Sedan skapades funktionerna ”USART2_read” och ”USART2_write”. Dessa användes för att skriva till och läsa från USART2-modulen. Write-funktionen väntar tills sändbufferten är tom, skickar en karaktär genom USART2 och returnerar parametern. Read-funktionen väntar tills en karaktär har mottagits, hämtar karaktären från USART2 och returnerar den.

För att kunna utge data i terminalen gjordes en struct för att strukturera överföringsströmmarna. Sedan gjordes två funktioner för att kunna läsa och skriva från standardinmatningen (terminalen) och

avslutningsvis gjordes en setup-funktion för att vid behov kunna testa läs- och skrivfunktionerna i terminalen.

3.1.3 LED.h

Headerfil för periferi-drivrutinen. I denna fil definierades nödvändiga makron, enums och structs för att bland annat specificera attribut för färg och status hos LED-lampan. Filens huvudsakliga delar bestod av:

- Makro för att definiera vilken GPIO-port (B) som skulle vara ansvarig för LED-funktionen.
- Makro för att definiera klocksignalen för porten.
- Makro för att definiera bitpositionerna för de GPIO-pins som va kopplade till LED-lamporna och sedan mode-bits för varje LED-färg.
- Två enum-typer, ”LedColor_Type” och ”LedState_Type”, för att representera LED-färgerna och dess status (av/på).
- En struct med två medlemmar (av enum-typerna) för att definiera de attribut en LED-lampa kunde bestå utav.
- Deklaration av funktionen för LED-konstruktorn och funktionera för kontroll och uppdatering av LED-lampans status.

3.1.4 LED.c

Denna fil börjar med att definiera konstruktor-funktionen ”Led_ctor”, som initierade LED-objektet med en viss färg och status. Först sätter konstruktorn färgen och statusen för LED-lampan genom att använda pekaren "me" och variablerna "_color" och "_state". Därefter aktiveras klockan för den port som LED:en är kopplad till (LED_PORT_CLOCK) genom att ändra värdet i register RCC->AHB1ENR. Sedan användes en switch-sats för att välja rätt portkonfiguration beroende på LED-färgen (röd, gul, blå, grön). Om färgen matchade och statusen var i påslaget tillstånd sattes LED:en på, annars slogs den av.

Sedan implementerades ytterligare två funktioner i denna fil: ”Led_setSate” och ”Led_getState”. Dessa användes för att ställa in och kontrollera statusen på LED-lampan och implementerades med switch-satser, i likhet med konstruktorn, för att avgöra hur LED:en skulle manipuleras.

3.1.5 Main.c

Programmets startpunkt vid exekvering. Här konstruerar och hanterar vi LED-lampan genom att anropa drivrutinernas funktioner. Vi börjar med att anropa ”USART2_init” för att upprätta kommunikationen. Sedan konstruera vi två LED-lampor (led1, led2) med rött resp. blått ljus och slår på led2 men inte led1 med hjälp av konstruktorn från LED.c. Därefter hämtar vi status för led1 och avslutar main-funktionen med att stänga av led2.

3.1.6 stm32f4xx.h

Enhetsspecifik fil för angivelser om stm-hårdvaran. Användes för att definiera klocksignalen för porten och mode-bitar för varje LED-färg.

3.2 Git/Github

Git och github användes i samspel för att underlätta hanteringen av projektet. Delvis för att göra det lättare att spåra ändringar i källkoden (git) men också för att lagra och dela detta projekt (gihub). En filstruktur enligt projektanvisningarna skapades först lokalt upp på datorn där sedan ett git repository initierades och länkades med ett nytt remote repository på github. Filstrukturen bestod av:

- Huvudmapp med en README-fil för övergripande beskrivning av projektets innehåll
  …samt tillhörande undermappar för:
- Hårdvarumapp – Dokumentation avseende STM-plattformen.
- Källkodmapp – Alla kodfiler (.c .h) till programvaran.
- Rapportmapp – Textbaserad rapport för att redovisa och mer ingående beskriva arbetet.

Git användes flitigt från terminalen för att få in övning och vana på att jobba med git via dess textbaserade kommandon. Alla delar från uppsättning av lokal filstruktur, arbete med git och uppladdning till github sköttes via terminalen. Totalt sett upplever jag att detta projekt har givit bra övning och lärande i många väsentliga delar gällande mjukvaruutveckling för inbyggda system och kommer troligtvis ha nytta av det i en fortsatt karriär inom IT.

4. Källförteckning

- *Muntlig information och föreläsningsunderlag från utbildaren Ludwig Simonsson*
- https://www.codrey.com/embedded-systems/uart-serial-communication-rs232/
- https://deepbluembedded.com/stm32-usart-uart-tutorial/
- https://www.rapidtables.com/convert/number/binary-to-hex.html
- https://www.st.com/resource/en/reference_manual/dm00119316-stm32f411xc-e-advanced-arm-based-32-bit-mcus-stmicroelectronics.pdf
- https://www.st.com/resource/en/datasheet/stm32f411re.pdf
- https://www.st.com/resource/en/user_manual/um1724-stm32-nucleo64-boards-mb1136-stmicroelectronics.pdf
- https://www.youtube.com/watch?v=dnfuNT1dPiM