NACKADEMIN

Inbyggda system: Arkitektur & Design

Student: Meriem Mohammed
Email: Meriem.mohammed@yh.nackademin.se

Innehållsförteckning

Introduktion

Inom den moderna teknikvärlden är en välutvecklad kommunikation mellan olika enheter och system väsentlig, detta för att möjliggöra pålitlig dataöverföring med kommunikationsprotokoll. UART (Universal Asynchronous Receiver-Transmitter) är ett exempel på ett protokoll för seriell datakommunikation.

Denna uppsats fokuserar på utvecklingen av en drivrutin för UART på stm324f11x-plattformen, som är en mikrokontroll plattform med funktioner som kan prestera för inbyggda system applikationer.

Målet med denna uppsats är att undersöka de olika aspekterna av att utveckla en UART-drivrutin för stm324f11x-plattformen, som tillsammans med en LED-applikation används för att styra en LED. Detta görs med hjälp av STM datablad som innehåller information om UART-protokollets funktioner samt hårdvaru arkitekturen.

I den återstående delen av denna dokumentation kommer vi att gå in på detaljerna i utvecklingsprocessen för UART-drivrutiner och hur de tillämpas på stm324f11x-plattformen.

Utvecklings Dagbok

Dag 1:

Dagens arbete började med att skapa och strukturera upp mappar, dels för att enklare kunna navigera bland filerna under projektets gång, men även för andra som ska ta del av projektet.

Det är möjligt att skriva all kod i en och samma fil, dock hade det blivit mycket och överväldigande för ögat. Av den orsaken är koden separerad i mindre delar med headerfiler och källkod.

I **UART.h** (header-fil med viktiga funktioner) definieras makro och en ***#ifndef*** som ska säkerställa om den specifika makron har definierats. Det sker en initialisering av vår testfunktion, som skapas i källkoden, men även vårt kommunikationsprotokoll UART. Den initialiseras med 'USART' som går att användas med UART funktionerna. USART gör, till skillnad från UART, det möjligt att programmera både synkront (kommunikation i realtid) och asynkront (icke - realtid). Kommunikatören konverterar parallel data till seriell data. [1]

Dag 2:

Eftersom mikrokontrollers är designade för att kontrollera andra enheter[2] , började jag dagen med att läsa igenom kurslitteratur för att få djupare förståelse över hur mikrokontroller fungerar i allmänhet. Eftersom vi hanterar portar, input/outputs och kommunikation mellan två olika komponenter tyckte jag att det verkade som en rimlig start för dagen innan jag gick vidare till databladet.

Jag fortsatte sedan med **UART.cpp.** Där aktiveras klockan, som är nyckel till kommunikationen, genom att följa stm324f11x - registret och dess instruktioner. Pilen i koden pekar nämligen till STM-registret. RCC (Reset and clock control) är en motor som har hand om olika delar av mikrokontrollern, som exempelvis olika bussar, processorn och andra hårdvarukomponenter. [3]

Med RCC har vi aktiverat klock tillgången genom att följa anvisningarna i registret som säger att bit 17 måste sättas till 1. Detta görs med hjälp av hexadecimala siffror för att få det motsvarande binära talet 1. Vi sätter även bit 0 till 1 för att aktivera klock tillgång till port A för GPIO (General-Purpose input/output) och detta för att kunna kontrollera signaler och strukturer.[4]

---

[1] Embedded Communication Protocols and Internet Of Things
[2] First steps with embedded systems, Byte Craft, s. 7
[3] STM32H7x3/x5/x7 - RCC, STMicro, s. 2
[4] STM32 datasheet, STMicro, s. 118

2 stycken pins aktiveras som har anknytning till port A för att UART kommunikationen ska fungera. De har möjlighet att anta alternativa funktioner, det vill säga att de antingen exempelvis kan fungera som en timer eller en pin för något protokoll. Alternativa funktioner används då en tvåvägskommunikation ska initialiseras. Hädanefter definierar vi vilket mode respektive pins ska anta (Input/Output).

Dag 3:

Dagens mål var att bli färdig med **UART.cpp**-filen**.** Fortsättningsvis går vi vidare i koden till uppbyggnaden av UART för att mer detaljerat kunna kontrollera kommunikationen. Baudrate har att göra med hastigheten som informationen överförs med. Här sätts vår Baud rate till 9600 bitar per sekund. Ett av kontroll registret används för att sätta igång sändaren och mottagaren för informationen och 2 av kontroll registret nollställs.

Deklaration av en skriv - och läs funktionen skapas. **USART2_write** används för att skicka data genom UART. Den väntar på att sändningsregistret ska bli tomt för att kunna skicka en byte till dataregistret. **USART2_read** används för att ta emot data via UART. Den väntar på att det ska finnas data i dataregistret för att kunna returnera den mottagna datan.

Dag 4:

Fokuset denna dag låg på LED-filerna. Vi skapar en **led.cpp-fil** och i den skapar vi konstruktorn, med 2 parametrar, som ska ansvara och initiera LED objekten med färg och status (on/off). Sedan aktiverar vi klockan för LED-porten genom att sätta in rätt bit enligt registret. Därefter används en switch sats för att sammanställa LED-porten och ställa in utgångar för den angivna färgen av lampan. En funktion som ska användas för att ändra statusen för lampan skapas och döps till ***setState().***

Vi har även skapat ytterligare en switch sats som ska kontrollera den aktuella färgen för lampan och ställa in rätt läge. Slutligen har vi ***getState()*** som ska hämta status för den aktuella lampan och hanterar den efterfrågade färgen och returnerar dess status.

I **led.h-fil** inkluderas standardbiblioteket för programspråket men även två header filer: UART och stm32f4xx för tillgång till definitioner och funktioner. I denna fil definieras:

- GPIOB som ansvarig för LED-funktionen
- Klocksignalen för GPIOB-porten
- 4 färger för LED-pinnar genom att tilldela dem pin nummer och bit positioner

Vi skapar två stycken uppräkningstyper med namnet LedColor_Type och LedState_Type som ska representera både färgerna och statusen som kan antas. En LED klass definieras med 2 privata attribut och en konstruktor som tar emot just färgtypen och statusen. Klassen har även två funktioner, **setState** (ändrar statusen för LED-färg) och **getState**(hämtar status för färgen).

Dag 5:

Dagens fokus låg på main.cpp som är huvudprogrammet. Här sker deklaration av 3 variabler som ska lagra och hålla koll på LED-lampornas tillstånd och status. I huvudfunktionen initieras **UART2**. Vi skapar 3 LED objekt som initialiseras med en färg och vilket tillstånd respektive LED har(on/off). LED-tillståndet för led1 hämtas och lagras i variabeln led_state. Minnet frigörs genom att använda ‘delete’- operatorn och radera led3 objektet. I slutet av filen skapas en loop som gör att programmet körs i oändlighet tills den bryts av ett ‘break’ påstående.

Slutresultat

https://github.com/gaia95/UART_Driver_Development

Källhänvisning

Ludwig Simonsson, Lektion Inspelningar

First steps with embedded systems, Byte Craft
https://studentportal.nackademin.se/pluginfile.php/319462/mod_resource/content/1/Bok1.pdf

Datasheet för hantering av Microcontrollers: STM32F405xx , STMicro
https://www.st.com/resource/en/datasheet/dm00037051.pdf

Datasheet for STM32H7x3/x5/x7 - Reset and clock control, STMicro
STM32H7-System-Reset and clock control (RCC)
file:///Users/meriem/Downloads/STM32F411x%20Reference%20Manual.pdf

Embedded Communication Protocols and Internet Of Things
http://www.gpcet.ac.in/wp-content/uploads/2018/08/UNIT-V.pdf