NACKADEMIN

Inbyggda system: arkitektur och design

Individuell inlämningsuppgift

Fredrik Jonsson – Fredrik.Jonsson@yh.nackademin.se

Github repository: https://github.com/FreJons/Inbyggda_System.git

Innehåll

Inledning

Universal Asynchronous Receiver/Transmitter (UART) är ett kommunikationsprotokoll som används för seriell dataöverföring mellan två enheter. UART-protokollet har funnits sedan 60-talet och har varit en viktig del av kommunikationen mellan datorer och externa enheter, såsom modem, sensorer och andra enheter. Även om det finns andra kommunikationsprotokoll, såsom USB och Ethernet, är UART fortfarande viktigt på grund av enkelheten och pålitligheten som protokollet tillför.

UART-protokollet är vanligtvis implementerat som hårdvara på mikrokontroller och andra integrerade system. Detta innebär att UART kan kommunicera med andra enheter utan behov av speciell programvara eller drivrutiner. UART kan även anslutas till en mängd olika enheter och används ofta i system som kräver enkel och pålitlig seriell kommunikation.

Detta projekt gick ut på att få till ett fungerande program som hanterar en LED. Med hjälp av dokumentation tillhörande STM32-miljön kunde man sätta upp en drivrutin för hur kommunikationen skulle ske. Vidare följde kod för hanteringen av funktionerna för LED.

Genomförande

I början av kursen fokuserade vi mycket på att gå igenom de vanligaste kommunikationsprotokollen som finns och vad som kännetecknar dem. Föreläsningarna handlade mycket om hur kommunikationen sker och vad alla olika begrepp innebär.

Då vi skulle använda oss av UART i vårt projekt lades mest fokus på detta. Vi fick därför prova att kommentera ut en drivrutinskod där UART konfigurerades, till vår hjälp hade vi dokumentation där all information gällande STM32 plattformen och dess konfigurationsmöjligheter fanns.

Vi skulle, som ett inledande steg i projektets början, ordna en god mappstruktur som vi sedan skulle pusha till vårt repository på Github. Här ska alla beståndsdelar av projektet ligga. Källkod, rapport, dokumentation och en readme. Efter detta började jag gå igenom och försöka förstå de kodfiler vi skulle jobba med och började med arbetet att kommentera dessa med hjälp av dokumentationen. Jag tyckte det var svårt att greppa vissa koncept med bland annat de hexadecimala värdena som till exempel GPIO pinsen sattes till.

Efter ett kort samtal med läraren om detta framgick det att man först bör konvertera dem till binära tal för att kunna utläsa vilken bit som motsvarar vilken pin i tabellen i dokumentationen. Denna vetskap gjorde det mycket enklare att förstå dokumentationen fullt ut då tidigare försök resulterat i att man hittat rätt avsnitt/sida men sedan fastnat där pga svårigheter att tyda tabellen som presenteras.

Resultat

De fem filer jag arbetade med är: UART.h, UART.cpp, LED.h, LED.cpp & Main.cpp. Nedan följer en förklaring på vad dessa filer gör och deras relation till varandra:

UART.h

Definierar en header-fil för UART-kommunikation på STM32F4-mikrokontrollern. Den innehåller funktionsdeklarationerna “USART2_Init” och “test_setup”, samt inkluderar stm32f4xx.h och stdio.h biblioteken. Funktionen “test_setup” används inte i detta fall. Den har också en "include guard" för att säkerställa att filen inte inkluderas mer än en gång under kompileringen.

UART.cpp

Här konfigureras UART kommunikationen och hur denna ska ske.

I den första raden inkluderas en headerfil "uart.h" som innehåller UART-funktioner. Sedan definieras en funktion "USART2_Init" som initierar UART2. Inuti denna funktion sätts klocktillgången för UART2 och port A, GPIO-pinnarna förbereds för alternativ funktion genom att rensa och sätta specifika bits i "GPIOA->MODER" och "GPIOA->AFR[0]", baudhastigheten sätts till 9600 bps och datastorleken sätts till 8 bitar, ingen paritet och 1 stoppbit. Till sist aktiveras UART genom att sätta en bit i "USART2->CR1".

Det finns också två funktioner för att skriva och läsa data från UART: "USART2_write" och "USART2_read". I "USART2_write" väntar programmet tills det finns tillgängligt utrymme för att skicka data och sedan skickas data till dataregistret. I "USART2_read" väntar programmet tills det finns data att läsa och hämtar sedan datan från dataregistret.

Sammanfattningsvis initierar "USART2_Init" UART2 med specifika inställningar, medan "USART2_write" och "USART2_read" används för att skicka och ta emot data från UART.

LED.h

Header-fil som definierar konstanter, typer och en klass som används för att styra vår LED på STM32F4-mikrokontrollern. Header-filen inkluderar också andra header-filer som behövs.

Koden börjar med en header guard, som är en mekanism som förhindrar dubbelinkludering av header-filer likt include guard i uart.h.

Därefter inkluderas tre header-filer: stdint.h, uart.h och stm32f4xx.h.

Stdint.h innehåller definitioner av typer med specifik storlek, uart.h innehåller deklarationer för funktioner som används för kommunikation via UART och stm32f4xx.h innehåller enhetsspecifika angivelser om hårdvaran för STM32F4.

Sedan definieras flera konstanter för att ange vilken GPIO-port och vilka pinnar som används för att styra LED:en. Konstanterna används senare i koden för att sätta och läsa från GPIO-porten och dess pinnar.

Efter detta definieras två enum-typer: LedColor_Type och LedState_Type. LedColor_Type definierar de olika färgerna som LED:en kan ha (röd, grön, gul och blå) och LedState_Type definierar de två lägena som LED:en kan ha (av eller på).

Slutligen definieras en klass som heter Led med två privata attribut, color och state, och två publika funktioner, setState och getState. Konstruktorn för klassen tar två argument, _color och _state, som används för att initialisera de privata attributen. Funktionen setState används för att sätta LED:ens tillstånd till antingen på eller av och getState används för att hämta LED:ens nuvarande tillstånd.

I slutet av header-filen avslutas header guarden.

LED.cpp

Den här koden är en del av en klass som hanterar LED-lamporna på mikrokontrollern. Klassen innehåller funktioner som styr att tända och släcka LED-lampor.

I början av koden definieras konstruktorn för klassen, vilket är en funktion som körs när en instans av klassen skapas. Konstruktorn tar två parametrar som anger LED-lampans färg och dess nuvarande status (tänd eller släckt).

Först sätter konstruktorn instansvariablerna i klassen till de värden som skickats in som parametrar. Därefter aktiveras klockan för GPIO-porten som LED-lamporna är kopplade till.

Sedan används en switch-sats för att konfigurera GPIO-pinnarna för den valda LED-lampan baserat på dess färg och status. Om statusen är ON (tänd), sätts GPIO-portens output-pinnar till hög för att tända LED-lampan. Annars sätts pinnarna till låg för att släcka LED-lampan.

Funktionen setState() används för att ändra statusen på LED-lampan efter att den har skapats. Här används också en switch-sats för att välja rätt LED-lampa, och GPIO-portens output-pinnar ändras beroende på önskad status.

Main.cpp

I main.cpp körs koden med alla de funktioner och klasser som finns i de andra filerna. Först inkluderas filen "led.h" som innehåller klassdefinitionen.

Sedan skapas tre variabler av typen LedState_Type för att hålla statusen på tre olika LED-lampor. Därefter skapas en instans av klassen Led med namnet led1 och definieras som en röd LED-lampa i läge ON.

I huvudfunktionen som heter main initieras USART2 och sedan skapas en annan instans, (led2), av klassen Led med två inparametrar av typen LedColor_Type för definition av färg, (blå), och LedState_Type som sätter LED i läge ON

En pekare till en instans av klassen Led med namnet led3 skapas också och definieras som en gul LED-lampa i läge ON.

Sedan hämtar koden statusen för led1 genom att använda funktionen getState() i Led.cpp och lagrar den i variabeln led1_state.

Därefter sätts statusen på led1 till OFF genom att använda funktionen setState() i Led.cpp. led3 tas bort och minnet som användes för det frigörs med delete.

Till sist finns en evighetsloop som gör att programmet fortsätter att köras utan att avslutas.

Källhänvisning

- Material från kursen och muntlig information från Ludwig Simonsson
- Datasheet stm32f411re
- Reference Manual för stm32-plattformen
- User Manual för stm32-plattformen