Introduktion:

Denna akademiska rapport beskriver utvecklingen av en drivrutin för UART-kommunikation på STM32F411x-plattformen. Syftet med rapporten är att beskriva processen för att utveckla och implementera en pålitlig och effektiv drivrutin för att hantera kommunikationen mellan mikrokontrollern och andra enheter som använder UART-protokollet.

Målet med rapporten är att erbjuda en grundlig beskrivning av designprocessen för en högkvalitativ drivrutin, samt att demonstrera hur denna drivrutin kan användas i praktiska tillämpningar på STM32F411x-plattformen.

Rapporten kommer att innehålla en detaljerad beskrivning av utvecklingsprocessen, inklusive hur drivrutinen har utformats och implementerats på plattformen. Vi kommer också att beskriva hur drivrutinen testades och utvärderades för att säkerställa att den möter kraven på tillförlitlighet och effektivitet.

Följande avsnitt kommer att beskriva plattformen, dess egenskaper och begränsningar samt ge en översikt över de olika delarna i den utvecklade drivrutinen. Vi kommer också att beskriva hur vi testade och utvärderade drivrutinen, och presentera resultaten av dessa tester.

Genom att läsa denna rapport kan läsaren förvänta sig att få en grundlig förståelse av utvecklingsprocessen för en drivrutin för UART-kommunikation på STM32F411x-plattformen.

Utvecklingsdagbok:

Dag 1:

Under dagens arbete var fokus på att bekanta sig med datablad och annan dokumentation för projektet med att utveckla en drivrutin för UART-kommunikation på STM32F411x-plattformen.

Dagen började med att jag tog mig igenom den tekniska dokumentationen för STM32F411x-plattformen och UART-protokollet för att få en bättre förståelse för dess egenskaper och begränsningar. Jag lade särskild tonvikt på att förstå hur UART-protokollet fungerar och hur det kan integreras med STM32F411x-plattformen.

Jag ägnade sedan resten av dagen åt att läsa igenom datablad för de olika komponenterna på STM32F411x-plattformen, inklusive UART-modulen. Jag gick igenom varje register och beskrivning av dess funktioner för att få en bättre förståelse av hur jag skulle kunna använda dem i min drivrutin.

Under dagen stötte jag på några utmaningar och frågor som jag behövde ta itu med. Till exempel stötte jag på en del oklarheter i dokumentationen om vissa funktioner i UART-modulen och behövde ta hjälp av mina klasskamrater och andra tekniska källor för att få en klarare förståelse av dessa funktioner.

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 utvecklingsprocessen.

Dag 2:

Dag två av projektet fokuserade jag på att utveckla själva drivrutinen för UART på STM32F411x-plattformen. För att göra detta använde jag mig av den samlade filen "UART.h" som innehåller alla headers och funktioner som behövs för att implementera UART-protokollet.

Först skrev jag en funktion som heter "USART2_Init" som initierar USART-protokollet och dess beståndsdelar. Detta innebär att jag aktiverade klocktillgången för UART2 genom att sätta bit 17 i APB1ENR till 1. Jag aktiverade också GPIOA genom att sätta bit 0 i AHB1ENR till 1. Därefter valde jag alternativ funktion för de valda pinsen PA2 och PA3 och valde typen av alternativ funktion för de valda pinsen. Jag gjorde detta genom att rensa bitarna 4-7 och 8-15 för att förbereda pins PA2 och PA3, sätta bitarna 5 och 7 till 1 för att aktivera alternativ funktionalitet på pins PA2 och PA3, och sätta bitarna 8-11 samt 12-15 till formatet 0111.

Efter att ha konstruerat enhetens kommunikation på detta sätt konfigurerade jag UART:en genom att sätta standard baud-rate till 9600bps med hjälp av hexadecimalen 0x0683. Jag satt också tx och rx till att arbeta i 8 bitars data, 1 stopp-bit och ingen paritet, genom att sätta bit 3 och bit 2 i CR1 till 1. Jag nollställde sedan CR2 och CR3 och omställde bit 13 (UART-aktiveringen) till 1 för att aktivera UART:en.

Sedan skrev jag funktionerna "USART2_write" och "USART2_read". "USART2_write" är 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. Jag använde

bit 7 i SR för att kontrollera statusen. Jag satte också överföringen av byten till dataregistret genom att sätta ch & 0xFF i DR.

"USART2_read" är en läsfunktion som läser in data som skickas till mikrokontrollern. Jag använde bit 5 i SR för att kontrollera om det finns mer data att hämta och returnerade sedan datan från DR.

Totalt sett tog det ungefär en dag att utveckla själva drivrutinen för UART på STM32F411x-plattformen, men det var en viktig del av projektet och jag var nöjd med resultatet.

Dag 3:

Dag tre av arbetet fokuserade på att utveckla en periferi-drivrutin som skulle fungera som ett komplement till UART-drivrutinen. Koden för periferi-drivrutinen var implementerad som en klass, Led, som innehöll attribut för färg och tillstånd för LED-lampan.

För att kunna använda klassen behövde vi definiera vissa pins och mode-bitar för varje färg som användes av LED-lampan. Dessa definierades med hjälp av fördefinierade makron, t.ex. #define LED_RED_PIN (1U<<14). Vi definierade också vilken GPIO-port som skulle användas för LED-funktionen och klocksignalen för porten.

Vi använde en enhetsspecifik headerfil, stm32f4xx.h, för att ange enhetsspecifika inställningar om hårdvaran. Denna fil användes för att definiera klocksignalen för porten och mode-bitar för varje LED-färg.

Vi definierade också två enum-typer, LedColor_Type och LedState_Type, för att beskriva vilka färger som kunde användas och vilka lägen som LED-lampan kunde vara i.

Sedan implementerades Led-klassen med en konstruktor som tog in färg och tillstånd för LED-lampan som parametrar. Klassen hade också två funktioner, setState och getState, för att ställa in och hämta tillståndet för LED-lampan.

Den utvecklade periferi-drivrutinen skulle användas för att styra en LED-lampa och skulle kunna användas tillsammans med UART-drivrutinen för att skicka och ta emot data.

Dag 4:

Dag fyra av arbetet med projektet var fokuserat på att strukturera dokumentationen och skapa ett Github repo för att göra det lättare att organisera och dela information om projektet.

Först skapade jag ett nytt Github-konto och skapade sedan ett nytt repo med lämpligt namn och beskrivning. Därefter laddade jag upp all befintlig kod till repo:t och skapade en README-fil som innehöll en kort beskrivning av projektet, dess syfte och en lista över de komponenter och enheter som användes.

Nästa steg var att skapa en struktur för dokumentationen och definiera vilka typer av dokument som skulle ingå. Jag följde här den försatta guiden som utbildaren tilldelade oss. För varje område skapade jag en separat mapp i repo:t och lade till relevanta dokument.

För att göra dokumentationen mer lättillgänglig och lättförståelig la jag till en beskrivning av varje dokument och dess syfte i README-filen, samt länkar till de relevanta filerna.

Efter att ha strukturerat dokumentationen och laddat upp all befintlig kod på Github kunde utbildaren enkelt få tillgång till all information som de behövde för att inleda bedömningen.

Detta minskade risken för att information skulle gå förlorad eller förbises och underlättade samarbetet mellan elev och utbildare.