NACKADEMIN

Pontus Söderlund - pontus.soderlund@yh.nackademin.se

Inledning
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.

DAGBOK
/* Glömde hur det var som man gjorde för att testa sin egna utveckling så i slutändan så skickas det istället in de existerande dataprogrammen men med min tolkning på hur de fungerar samt varför. Jag gjorde detta för att undvika att skicka in något som inte fungerar även fast det jag gjorde inte var så väldigt annorlunda från ditt. Men detta var det jag dokumenterade tidigare..
*/

Metod: För att lära mig om UART-kommunikation började jag med att läsa om dess grundläggande funktion och fördelar. Jag tittade på olika applikationer där UART-kommunikation

används, såsom dataöverföring mellan datorer och enheter, till exempel microcontrollers. Jag studerade också faktablad om UART-komponenter, särskilt om UART-hårdvaran och dess olika register.

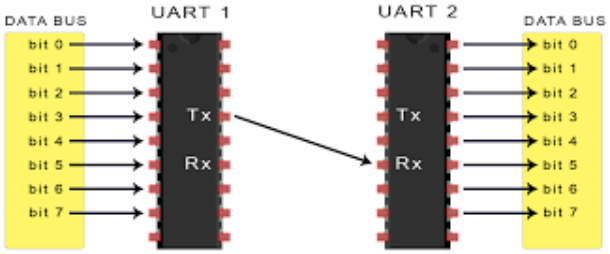
För att göra en UART-drivrutin, började jag med att bestämma vilken av mikrokontrollerpinarna som skulle användas för att ansluta till UART-hårdvaran. Jag använde sedan datasheet och faktablad för att lära mig om UART-hårdvarans olika register och hur de används för att initiera och konfigurera UART-kommunikationen.

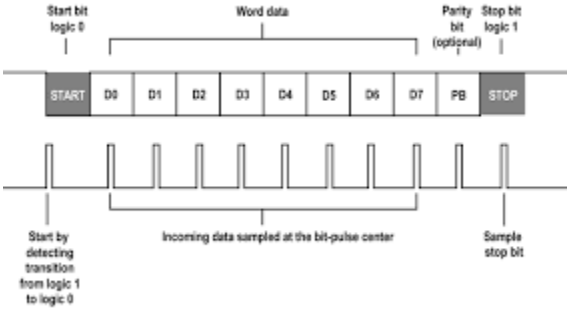
Efter att ha förstått lite mer av hårdvaran och dess register, skapade jag en grundläggande kod för att initiera UART-kommunikationen. Jag skrev sedan kod för att sätta upp överföringshastighet, datastorlek och stoppbitar.

Kod

Uart

UART används för att skicka och ta emot data mellan två enheter över en seriell kommunikationskanal. Detta är användbart i många olika applikationer där enheter måste kommunicera med varandra på ett pålitligt och effektivt sätt till exempel i kommunikation mellan datorer och elektronik.





Led.cpp

Det hära data programmet är en del av en implementation av en LED klass som styr olika LED lampor på en stm32f4xx. Programmet inkluderar en headerfil LED.h som innehåller definitioner av LED beteckningar och funktioner som används.

Klassen har en konstruktor som tar två argument: en LedColor_Type som anger LED:ens färg och en LedState_Type som anger LED:ens start status. Konstruktorn sätter LED:ens färg och status och konfigurerar LED pinnarna för att sätta dem i rätt läge baserat på deras färg och status.

Klassen har också två andra funktioner: en för att sätta LED:ens status setState och en för att hämta LED:ens status getState. Funktionen setState sätter LED:ens status och uppdaterar LED pinnarnas output för att vara den nya statusen. Funktionen getState returnerar LED:ens status för en specifik färg.

Programmet använder en bitwise operation för att sätta och rensa bitar i LED_PORT->MODER och LED_PORT->ODR som är kopplade till LED-pinnarna på stm32f4xx. Dessutom använder programmet switch sats för att kontrollera LED:ens färg och välja rätt LED pin och register att manipulera. Programmet förväntar sig att de olika LED-beteckningarna och deras motsvarande portar och pinnar har definierats i LED.h filen.

Kod Exempel Led.cpp

Konstruktörsfunktion för en klass som heter Led, som tar två argument: _color och _state. Klassen Led representerar en LED på en hårdvaruenhet, och konstruktören används för att initialisera dens egenskaper.

Switch-satsen använder det inkommande _color-argumentet för att bestämma vilken LED-komponent som ska konfigureras. I detta fall _color är lika med RED.

För att sätta LED-portläget till output, används en bitwise|= för att ställa in LED_RED_MODE_BIT-biten i LED_PORT->MODER-registret.

Därefter kontrolleras LED:ens aktuella tillstånd för att avgöra om den ska vara på eller av. Om tillståndet är ON, används en bitwise |= för att sätta LED_RED_PIN-pinnen i LED_PORT->ODR-registret för att slå på LED:en. Om tillståndet är OFF används istället en bitwise och-operation &= för att nolla LED_RED_PIN-pinnen i LED_PORT->ODR-registret för att stänga av LED:en.

Main.cpp

Detta program inkluderar en headerfil led.h och definierar tre variabler av typen LedState_Type som heter led1_state, led2_state och led3_state. Sedan skapas ett objekt av klassen Led som heter led1 med röd färg och tillståndet ON.

I huvudfunktionen initieras USART2 och två nya LEDobjekt skapas, led2 med blå färg och tillståndet ON och led3 med gul färg och tillståndet ON. Först hämtas led1:s aktuella tillstånd och sätts till led1_state. Sedan ändras led1:s tillstånd till OFF. Slutligen tas led3 bort genom att kalla på delete operatören.
Programmet fastnar sedan i en evig while-loop.

Kod Inom Main.cpp

LedState_Type led1_state; LedState_Type led2_state; LedState_Type led3_state; deklarerar tre variabler av typen LedState_Type som kommer att användas för att lagra LED-lampornas tillstånd.

Led led1(RED,ON) skapar ett objekt av typen Led och initierar det med en röd färg och ON tillstånd. Detta objekt representerar den första LED-lampan.

led1_state = led1.getState(); hämtar LED1:s tillstånd med hjälp av getState() och tilldelar det till led1_state.

led1.setState(OFF); ställer om LED1 till OFF med hjälp av setState()-funktionen.

delete led3; frigör minnet som tidigare allokerades för led3-objektet.

Stm32f4xx.h

Headerfil med alla definition för alla register inom stm32f4xx mikrokontrollern. Headerfilen innehåller också definitioner av funktioner och konstanter som är specifika för stm32f4xx mikrokontrollern. Detta gör dett lätt att kommunicera samt styra enheter baserade på stm32f4xx.

Uart.cpp

är att sammanfatta, koden är en implementation av UART protokollet för att möjliggöra seriell kommunikation mellan en mikrokontroller och en annan enhet, t.ex. en dator och en mikrokontroller. Funktionerna USART2_write och USART2_read används för att skicka och ta emot data via UART protokollet.

Kod inom Uart.cpp

Koden initialiserar UART-enheten och sätter kommunikationens parametrar så som baudrate och datastorlek och definierar funktioner för att skicka och ta emot data med hjälp av UART.

Funktionen USART2_Init () påbörjar USART-protokollet och beståndsdelar genom att aktivera klock tillgång för UART2, input output och pins relaterade till vald port.

Den väljer också typen av alternativ funktion för de valda pinnarna. USART2_write () är en funktion som överför data till terminalen medan USART2_read () är en läsfunktion som tar emot information från vår mikrokontroller.

Referenser

https://github.com/ludwigsimonsson
https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter
https://www.circuitbasics.com/basics-uart-communication/
De länkade Uart datasheets