

**Inbyggda System Arkitektur och Design**

**Inlämning**

**Inledning…….………………………..………………………..………**

**Kod….………………………..………………………..…….…….……**

**Arbetsbeskrivning………………………..………………………..…**

Led.h…….…………...

Led.cpp…….…………

Uart.h………………….

Uart.cpp…….…………

Main.cpp……………...

**Referens & bilagor..………………………..…….…………….………**

IoT 2022

Ellära

Jonatan Ghirmay

[jonatan.ghirmay@yh.nackademin.se](mailto:jonatan.ghirmay@yh.nackademin.se)

**Inledning**

I denna rapport så kommer jag att gå igenom ett projekt för en LED kopplad till en STM32 mikrokontroller. Denna LED kommer att kommunicera med mikrokontrollern med hjälp av UART protokollet.

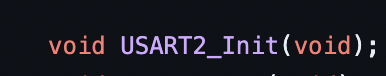
Syftet och målet med detta arbete är att påvisa kunskap och en övergripande förståelse inom “Inbyggda System” och i det här fallet genom detta LED projekt med STM32 mikrokontrollern.

I projektet så kommer det att finnas en kod “mapp” där all kod finns deklarerad med kommentarer som förklarar vad varje kod gör och har för funktion i kommunikationen mellan LED och mikrokontroller hårdvaran. I denna rapport så kommer jag främst att förklara koden och hur programmet för denna LED med STM32 mikrokontroller i UART-protokollet är uppbyggt och strukturerat.

Github länk: <https://github.com/jonghi11/UART_LED>

**Kod**

Filerna med kod är upplagda så att det finns en “Main” fil som initierar programmet och kallar på funktionen som skall sätta igång eller stänga av lampan på önskat sätt. Sedan så har vi två “Header” filer. I den ena “Headern” som är för LED så finns definitionerna för att enklare kunna läsa programmet samt vissa klasser och funktioner. Sedan så finns en “Header” fil för själva UART:en där vi främst har funktionen som initierar UART-protokollet inklusive en testfunktion, “STM32” mikrokontrollerns header inkluderad.



De två filerna med mest kod i är i detta fall LED och UART vardera “huvudfil” där vi i UART:ens huvudfil sätter pins, diverse register, portar till de värdena vi vill med hjälp av “bitwise operations". I LEDens huvudfil så har vi främst funktionerna för LEDen, där exempelvis ON/OFF funktioner finns samt switch cases med if-satser för hur LEDen med pins etc bör te sig när man skall byta lampa och liknande.

LED.h

Till en början så definierar vi porten på STM32 kortet som vi vill använda för LED funktionen. Där väljs "GPIOB"-porten som både kan användas för input och output. Sedan så definierar vi klocksignal för nämnda port genom att sätta den första biten som 1 och resten kvarstår som 0. Denna konstant sätts för att kontrollera klocksignal för LED-porten.





Efter detta så definieras bits som styr skall styra pins och moden till specifika LED färger.





I exemplen ovan så ser man hur vi definierat den 14:e biten från höger till 1 för att kontrollera röd pin och den 28:e biten från höger till 1 för att kontrollera moden på denna röda LED. Detta görs även sedan med resten av LED färgerna grön, gul blå på samma sätt med olika bits.

I koden så sätts sedan värdena med “enum” inkrementering för LED färgerna där varje färg har sitt eget värde från 0-3 och samma sak görs med ON = 0/OFF = 1 funktionen

RED = 0,

GREEN = 1

YELLOW = 2

BLUE = 3



Dessa kommer sedan att kunna kallas på från klassen “Led” med vardera argument för att kalla på attributen(color, state). I och med att dessa två attribut är privata så kommer vi göra en setter och getter funktion för att kunna kalla på dem i main filen.

LED.cpp

I LED filens huvudfil så börjar vi med att “include” LED headerfilen till programmet. Sedan så tar vi klassen “Led” från “Led.h” och för att sätta attribut, konfigurera i konstruktorn som vi skapat från klassen Led.



“RCC” som kontrollerar klocksignal på en på en lämplig frekvens för kärnan och “peripherals”. Den sätter vi till AHBENR1 som är ett register som har en högsta frekvens på 100 MHz. Dessa värden sätts med en OR bitwise i LED\_PORT\_CLOCK(GPIOB porten) för att enable:a klockan.

Vi har två switch cases

UART.h

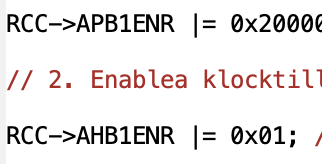
Kallar på och initiering

UART.cpp

Uart.h filen inkluderas till en början i UART filen med main en main.

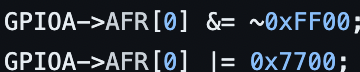
Sedan så har vi i en funktion med beståndsdelarna för UART-protokollet.

Vi enablar till en början klockan till UART med samt GPIOA porten .



Sedan så sätts med MODE registret bit 4-7 som inaktiva för pins 2 och 3 i port A. Detta för att förbereda pin 2 och 3.

Sedan sätts pin 5-7 bitsen till 1 som gör det möjligt för UART kommunikation via pins. 

Bitarna 8-15 inaktiveras för att på samma sätt förbereda för port A:s pin 2 och 3. Bitarna 8-11 sätts som 1 medans 12-15 sätts till 0. 

Därefter så sätts baud-raten i UART till det som har varit standarden, 9600 bits per sekund. Control register(CR) CR2 samt C3 nollställs medans CR1 till en början sätter tx(transmitter) och rx(reciever) till att arbeta med 8 bitars data. för att sedan också ställa om bit 13 i UART-aktiveringen till 1.

Avslutnignsvis så har vi en Write och en Read funktion som båda har busy wait loopar som körs när förhållandena stämmer. Alltså så fortgår programmet endast under rätt förhållanden. Detta kontrolleras med Status Register(SR) och Data Register(DR). I SR finns info om om nuvarande status på UART diverse felmeddelanden och om överföringar är klara etc.

DR har har ansvar för tx(transmitter) rx(receiver), alltså data in, ut och konfigurationen dem emellan. I “Write” funktionen så är DR satt så att funktionen har ett krav att statusen på överföringen är tom och kan ta emot nästa karaktär. Medan SR så har busy-loopen som endast låter programmet fortgå under rätt förhållanden.

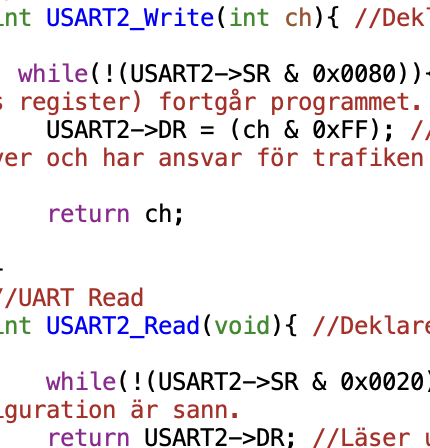
I Read funktionen så finns en busy-loop för SR som kontrollerar att förhållandena är korrekta för att programmet skall få fortgå och en return för DR som läser ut datan.

Main.cpp

Vi har till en början utanför ”main” tre stycken Ledstate\_type för tre LED. Detta är från led.h och kan ha värdena ON=1, OFF=0. Sedan så skapas led1 med argument för en röd lampa som skall vara ON. Den finns utanför mainkoden vilket gör att den är global. Sedan så har vi två instanser likt led1 som heter led2 och led3. Dessa två instanser finns innanför “main” vilket gör att dessa två inte är globala utan endast finns innanför main. Notera även att led3 är skapad som en pekare som pekar på “new” med värdena (YELLOW,ON). Denna pekare tas sedan bort senare i koden.

Setter och getter funktionerna kör sedan för led1 där gettern returnera statusen på led1 och settern sätter statusen som OFF på led1.

Sist så har vi en while loop som oändligt går utan att stänga ner programmet.



**Arbetsbeskrivning**

I detta projekt så har jag jobbat med koden och kommenterat i främst i EMACS samtidigt som jag förklarat koden i rapporten. Projektet klonades från github för att sedan läggas upp i min github. Började sedan medan en initial commit och push av filstruktur för att sätta igång arbetet. Därefter så har jag gjort commits och pushat upp framstegen med jämna mellanrum. Som ett exempel så har jag en commit som är “Led.h report” som jag sedan pushat upp. Den commit:en hänvisar till att jag gjort mest framsteg i rapporten Led.h kodförklaring under denna commit. Detta för att kunna hålla koll på vad som skulle försvinna om jag skulle vilja gå tillbaka till en tidigare commit registrerad och ha koll på vad som skulle försvinna i sådana fall.

Jag har även använt mig av de tre manualerna som vi fått för STM32 som för att förstå vad olika register har för funktioner och för att få en djupare förståelse för koden.

**REFERENS & BILAGOR**

