NACKADEMIN

Inbyggda System Arkitektur och Design Inlämning

IoT 2022
Ellära
Jonatan Ghirmay

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
Github HTTPS: https://github.com/jonghi11/UART_LED.git

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.h så finns definitionerna för att enklare kunna läsa programmet samt vissa klasser och funktioner. Sedan så finns en "Headerfil för själva UART:en där vi främst har funktionen som initierar UART-protokollet inklusive en testfunktion, “STM32” mikrokontrollersnebm header inkluderad.

```
void USART2_Init(void);
```

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.

```
#define LED_PORT GPIOB
```

```
#define LED_PORT_CLOCK (1U<<1)
```

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

```
#define LED_RED_PIN (1U<<14)
```

```
#define LED_RED_MODE_BIT (1U<<28)
```

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

```
private: //De två attribut
      LedColor_Type color;
      LedState_Type state;
```

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.

```
Led::Led(LedColor_Type _color, LedState_Type _state)
{

  this->color = _color;//color och state attributen t
  this->state = _state;//
```

“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. Den första har vi inuti Led- konstruktorn med argumenten för “color” och “state”. Vi sätter cases för de fyra färgerna och MODERn till output till de olika färgernas portar med ODR registret. Om “state” är “ON” så blir pins output aktiv annars så blir den inaktiv. Den andra switchen finns i getState(setter)-funktionen och ser till att rätt Led manipuleras. Avslutningsvis så finns en switch i getState(getter) funktionen som returnerar lampans status utav vald färg.

UART.h

Här finns initialiserings funktionen för UART samt en test funktonen.

UART.cpp

UARTt.h filen inkluderas till en början.
Sedan så har vi i en funktion med beståndsdelarna för UART-protokollet.
Vi enablar till en början klockan till UART med samt GPIOA porten .

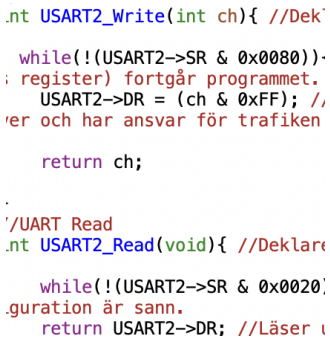
```
RCC->APB1ENR |= 0x20000
// 2. Enablea klocktill
RCC->AHB1ENR |= 0x01; /
```

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. `GPIOA->MODER &= ~0x00F0;`
Sedan sätts pin 5-7 bitsen till 1 som gör det möjligt för UART kommunikation via pins. `GPIOA->MODER |= 0x00A0;`

Bitarna 8-15 inaktiveras för att på samma sätt förbereda för port A:s pin 2 och 3.
Bitarna 8-11 sätts som 1 medan 12-15 sätts till 0.
```
GPIOA->AFR[0] &= ~0xFF00;
GPIOA->AFR[0] |= 0x7700;
```
Därefter så sätts baud-raten i UART till det som har varit standard, 9600 bits per sekund. Control register(CR) CR2 samt CR3 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 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 har en busy-loop 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 ”funktionen tre stycken Ledstate_type för tre LEDs. 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 main funktionen 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 och alltså förstörs utanför funktionen. Notera även att led3 har skapats 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.



```
.nt USART2_Write(int ch){ //Dek

  while(!(USART2->SR & 0x0080))
 register) fortgår programmet.
    USART2->DR = (ch & 0xFF); //
er och har ansvar för trafiken

    return ch;

.
'/UART Read
.nt USART2_Read(void){ //Deklar

    while(!(USART2->SR & 0x0020
.guration är sann.
    return USART2->DR; //Läser
```

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 och främst de olika registrena.

Med detta projekt så har så finns det mycket att ta med sig. Dels vilka saker som fungerat samt vilka saker som inte fungerat riktigt som man tänkt. Delarna som inte riktigt gått som man tänkt är det jag tar mest lärdom av.
Ett tag in på kursen så kände jag att det skulle bli naturligare att inte ha en dagbok denna gång i och med vissa förändringar och uppskjutningar och att det inte riktigt kändes naturligt med en dagbok. Detta beslut önskar jag att jag inte tagit då jag nu bättre förstår att en dagbok är tillför både framsteg som motgångar och för dokumenteringen av detta.
Jag har tidigare varit mer benägen att dokumentera framsteg.

.

REFERENS & BILAGOR

The clock controller provides a high degree of flexibility to the application in the choice of the external crystal or the oscillator to run the core and peripherals at the highest frequency and, guarantee the appropriate frequency for peripherals that need a specific clock like USB OTG FS, I2S and SDIO.

Several prescalers are used to configure the AHB frequency, the high-speed APB (APB2) and the low-speed APB (APB1) domains. The maximum frequency of the AHB domain is 100 MHz. The maximum allowed frequency of the high-speed APB2 domain is 100 MHz. The maximum allowed frequency of the low-speed APB1 domain is 50 MHz

6.3.9 RCC AHB1 peripheral clock enable register (RCC_AHB1ENR)

Address offset: 0x30

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									DMA2EN	DMA1EN	Reserved				
									rw	rw					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			CRCEN	Reserved				GPIOHEN	Reserved		GPIOEEN	GPIODEN	GPIOCEN	GPIOBEN	GPIOAEN
			rw					rw			rw	rw	rw	rw	rw

6.3.11 RCC APB1 peripheral clock enable register (RCC_APB1ENR)

Address offset: 0x40

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved			PWR EN	Reserved				I2C3 EN	I2C2 EN	I2C1 EN	Reserved			USART2 EN	Reser-ved
			rw					rw	rw	rw				rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3 EN	SPI2 EN	Reserved		WWDG EN	Reserved							TIM5 EN	TIM4 EN	TIM3 EN	TIM2 EN
rw	rw			rw								rw	rw	rw	rw

6.3.7 RCC APB1 peripheral reset register for (RCC_APB1RSTR)

Address offset: 0x20

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved			PWR RST	Reserved				I2C3 RST	I2C2 RST	I2C1 RST	Reserved			USART2 RST	Reser-ved
			rw					rw	rw	rw				rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3 RST	SPI2 RST	Reserved		WWDG RST	Reserved							TIM5 RST	TIM4 RST	TIM3 RST	TIM2 RST
rw	rw			rw								rw	rw	rw	rw

Bit 17 **USART2RST:** USART2 reset

Set and cleared by software.

0: does not reset USART2

1: resets USART2