Andreas Axelsson
Alexander Göransson
Viktor Mann

JÖNKÖPING UNIVERSITY
School of Engineering

2023-04-26

Enchipsdatorer – Laboration 6
PWM, Pulsbreddsmodulering

Målsättning

- Lära sig om avancerad funktionalitet hos en timer.
- Använda sig av en timers capture/compare-register för PWM-generering.
- Få många dioder att blinka fint, samtidigt.
- *(frivillig)* Splea upp ljud med en passiv summer.

Förberedelse

Föreläsningar om timers och PWM. Det rekommenderas att man läst kursboken enligt läsanvisningarna som finns i slutet på föreläsningsanteckningarna.

Examination

Resultat för laborationen presenteras muntligt för en laborationshandledare under laborationstillfället. Frågor i **fet stil** ska besvaras under redovisningen. Krav på kodstil måste följas.

Genomförande

Laborationen har 4 timmar handledd tid, men kan ta mer tid att genomföra, vilket då görs på egen hand.

Det är viktigt att kodningsarbetet sker individuellt. Det är okej att diskutera problemlösning med andra men det är absolut förbjudet att kopiera kod från andra. De lösningar som tas fram skall förses med bra kommentarer, korrekt indentering och bra variabelnamn. Koden ska m.a.o. vara lätt att läsa och förstå.

Hårdvara

- 1x Kopplingsdäck
- Kopplingskablar
- 1x RGB-LED
- 3x Motstånd (220Ω)
- *(frivillig)* 1x Passiv summer

Andreas Axelsson
Alexander Göransson
Viktor Mann

Enchipsdatorer – Laboration 6
PWM, Pulsbreddsmodulering

1. Förberedelse

I laborationens första del så ska de tre dioderna i en en RGB-LED dimmas med hjälp av en pulsbreddsmodulerade (pulse width modulation, PWM) signaler genererade av TIM1.

Vilka frekvenser brukar en LED-dimmer arbeta på?

Öppna *referensmanualen* och gör dessa steg:

1. Slå upp kapitlet för TIM1.
2. Titta på grafiken i figuren ”Advanced-control timer block diagram”.
3. Bli chockad över hur att opedagogiskt många saker som finns i den.
4. Så upp kapitlet för TIM9 till TIM11.
5. Titta på figuren ”General-purpose timer block diagram (TIM10/TIM11)”.
6. Ignorera de två rutorna nere till vänster i grafiken (input filter... och prescaler).

Skriv en kort beskrivning (2-4 meningar) av hur PWM-generering fungerar. Dessa måste nämnas i din beskrivning: ARR, CNT, CCR1[1], TIMx_CH1[2].

När man pratar om en PWM-signals ”duty cycle” så brukar den siffran vara angiven i procent. Så när du väljer värden så är det väldigt smidigt om ARR+1 blir 100, 1000 eller 10000. Se till att $T_{TIM1} = \frac{1}{f_{TIM1}} > 50\ \mu s$.

Vilka värden valde du för TIM1?

$f_{TIM1} =$

$p =$

$n =$

[1] I figuren är detta rutan ”Capture/Campare 1 register”.

[2] Att den heter CH1 är för att den är bunden till CCR1. TIM1->CCR2 skulle vara bunden till TIM1_CH2 osv.

2. Uppgifter

Ibland behöver man kunna styra signaler mer nyanserat än bara på/av. T.ex. så vill man kanske styra hastigheten på en motor eller dimma en lampa för lagom ljusnivå.

För att åstadkomma en mer gradvis varierande signal används ibland en Digital-to-Analog Converter (DAC). Istället för att avända en ”riktig DAC” så räcker det ibland med en signal som slår på/av väldigt fort. Istället för att justera en spänning upp och ner så justeras hur stor tidsandel (duty cycle) som signalen är hög under varje period. Signalen kan då ge sken av att vara på 30% om den endast är påslagen 30% av tiden.

Poängen med PWM är att den högfrekventa signalen filtreras så att dess skiftande inte märks. I fallet med LED så är det ögats begränsade uppdateringsfrekvens som står för filtreringen. För en PWM-signal till en motor så gör den högfrekventa på/av-signalen att den går lägre än maxhastigheten.

2.1. Uppgift 1 – Debugger och registermanipulering

Konfigurera CubeMX och anslut RGB-LED:en som beskrivs i kapitel 3.1.

Innan main-loopen ska TIM1 startas för de tre kanalerna du nyss konfigurerat. Den funktion du bör använda finns i `JU_STM32_HAL-beskrivning.pdf` i PWM-delen. Du måste anropa den tre gånger – en för varje kanal med PMW-signal.

Lämna main-loopen tom och starta din kod i debug-läge. Pausa körningen när TIM1 är påslagen och gå till SFRs fliken i debug-vyn.

Ändra värden i register CCR1, CCR2 och CCR3 medan körningen är pausad, notera att värdet i ARR är det högsta användbara.

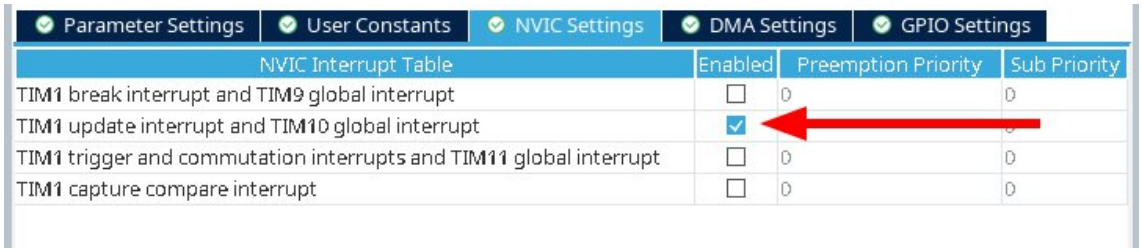
Varför kan du ändra ljusets intensitet när processorn är pausad?

Är de tre färgerna av samma ljusintensitet för samma värde på CCRx?

Om någon färg behöver högre duty cycle för att nå samma intensitet som den starkaste, vad är (grovt gissat) proportionerna?

2.2.Uppgift 2 - Färgcykler

Aktivera interrupts för TIM1. Den interrupt du ska ha är den som sker när räknaren slår runt (efter att CNT == ARR) och den heter ”update interrupt”:



För att interrupts ska börja genereras så är det den vanliga `...Base_Start_IT()` som ska anropas. Callback-funktionen är även den som vanligt, `...PeriodElapsed...()`.

För att testa så kan du börja med att ha

```
isr_count++;
```

som callback-funktion och sedan låta din main-loop ha denna kod:

```
uint32_t duty = (isr_count / 32) % 1000;
__HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, duty);
```

Modulo 1000 utförs då ARR == 999. Testa koden och verifiera att en av färgerna växlar intensitet med tiden.

För att få en mjuk färgförändring får du här större delen av koden som bör köras vid (*inte i!*) en callback. **OBS!** Denna kod är alldeles för långsam (c:a 30 mikrosekunder med $f_{CPU} = 84\ MHz$) för att ligga i en ISR.

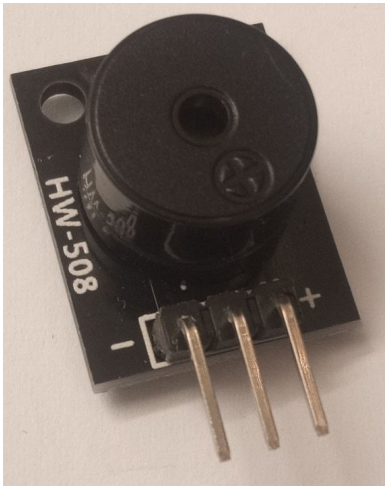
Dessa använder sig av `<math.h>`. Variablerna rf, gf och bf är av typen float. Missa inte att sätta half_arr!

```
static float t = 0.0f;
const  float offset = M_TWOPI / 3.0f;
const  float half_arr = 0.0f; /* set to (ARR+1) / 2.0f !!! */

rf = sinf((M_TWOPI * t * freq) + (0 * offset));
gf = sinf((M_TWOPI * t * freq) + (1 * offset));
bf = sinf((M_TWOPI * t * freq) + (2 * offset));
rf = (rf + 1.0) * half_arr;
gf = (gf + 1.0) * half_arr;
bf = (bf + 1.0) * half_arr;

t += 0.001; // should be (1 / f_TIM)

/* Code for setting PWM duty cycles goes here */
```

Gör variabeln freq (är mätt i Hz) global och sätt den till vad du tycker är lämpligt. Applicera vad du märkte om intensitet i uppgift 1 när du sätter DC-värden.

När din RGB-LED skiftar färger fint kan du redovisa. Bra jobbat och lycka till med projektet!

2.3.Uppgift 3 - Ljuduppspelning (Frivillig)



Ladda ner filen `samples.h` och inkludera i ditt projekt.

Filen innehåller två saker: siffran NUM_SAMPLES och en array med amplituddata, tänkt att spelas upp i en högtalare. Hela arrayen är tänkt att spelas upp med en samplerate på 16000 Hz. Detta är hur ofta du vill lägga ut en ny sample (skriva en ny duty cycle) på PWM-generatorn. Sätt alltså

$$f_{TIM} = 16\,000\ Hz$$

Då du vill kunna ha så detaljerad upplösning av samplingarna som möjligt så är det bäst om prescalern sätts till 0.

Välj en timer (ej TIM1) kapabel att generera en PWM-signal och aktivera den med beräknade värden. Glöm inte att aktivera interrupts för den.

Varje gång en interrupt sker tar du nästa värde från arrayen, skalar/justerar det och lägger det i CCR (med `__HAL_TIM_SET_COMPARE()`, som tidigare):

```
uint32_t duty = samples[sample_counter];
duty      += 16384;
duty      >>= 4;    // divide by 16
```

Koppla in den passiva summern (se bild, märk plustecken på ovansidan) mellan jord (minus-kontakten) och den port som din PWM-kanal ligger på (plus-kontakten).

Använd dig av dessa tre funktioner för att färdigställa din lösning:

```
HAL_TIM_PWM_Start()
HAL_TIM_Base_Start_IT()
void HAL_TIM_PeriodElapsedCallback(Tim_HandleTypeDef * htim)
```

Gör du rätt kommer du höra en en låtslinga.

Redovisning (men den var ju frivillig? (ja, jo)): Vad är det för låt?

3. CubeMX och kopplingar

Aktivera TIM1. Ställ in dess kanaler enligt bilderna och fyll i de värden du beräknat.

Ge även namn åt de portar som CH1, CH2 och CH3 läggs på. T.ex. kan de kallas `LED_R`, `LED_G`, `LED_B`.

Anslut motstånd på 220Ω mellan dessa GPIO-portar och LED-benen.