Laboration 3 i Datorteknik - Avbrottshantering

Laborationens mål är att få förståelse för avbrottshantering.

- Ladda hem filen Lab3.zip och packa upp på lämpligt ställe på ditt konto. Där i ligger färdiga projektfiler och källkod för uppgift1. Öppna projektet lab3.pew. Där finns en kodmall liknande den som beskrivs nedan

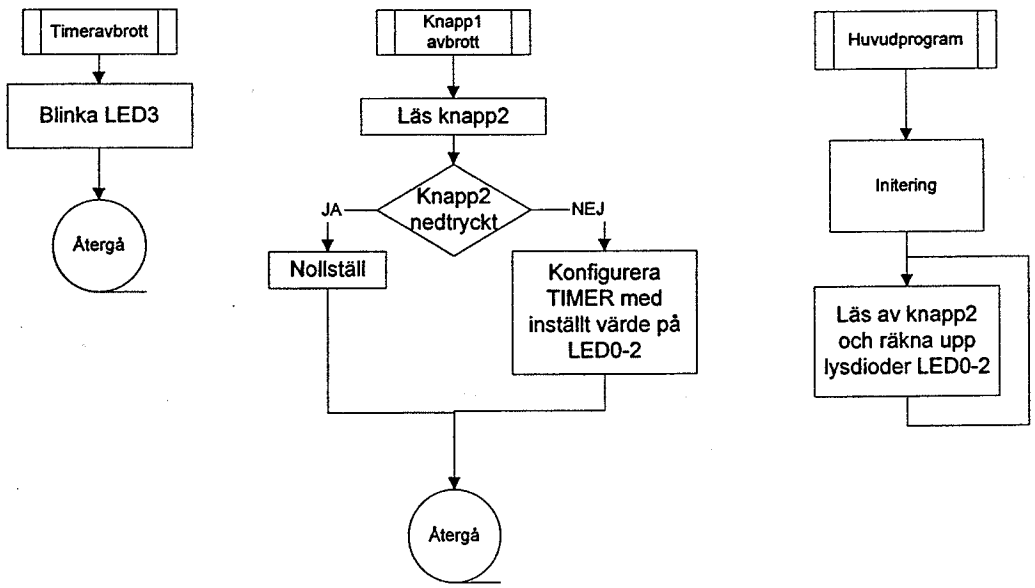
Målet med laborationen är att konstruera ett system enligt nedanstående.

Grundfunktion hos programmet

Laborationsplattformen har 4 lysdioder och 2 knappar. Dom kommer refereras till som LED3-LED0 samt knapp1 och knapp2.

LED3	LED2	LED2	LED0	K1	K2

- LED3 ska blinka med en viss frekvens.
- Frekvensen ska kunna ställas med knapp2. Då knapp2 trycks skall LED0-LED2 räkna upp binärt. Genom att trycka på knapp1 ska det binära tal som nu är inställt på LED0-LED2 förändra frekvensen på den blinkande LED3. Man ställer alltså in med knapp2 och bekräftar med knapp1
- Genom att hålla inne knapp2 samtidigt som man trycker knapp1 ska alla lampor nollställas samt systemet nollställas, dvs det ska vara som från början. En form av reset-funktion.
- Programmet ska använda timeravbrott samt avbrott från knapp1 och vara uppbyggt enligt nedanstående princip



Arbetsgång

Nedan beskrivs lämplig arbetsgång för programmet. Alla dom olika delarna beskrivs detaljerat i längre fram i olika kapitel.

1. Konfigurera avbrottskällor. Görs vanligtvis med registren INTMOD och INTMSK

2. Konfigurera stack för avbrott. Måste göras genom att gå över i IRQ-mode. Mode-övergång görs med CPSR.

3. Skriva en avbrottshanterare som läser av vilket avbrott som skett i INTPND-registret och hoppar till rätt avbrottskod. Avbrott kvitteras i INTPND.

4. Skriv avbrottskod för de avbrott man vill använda.

5. Anpassa vektortabellen till avbrottshanteraren som tar hand om IRQ-avbrottet.

6. Tillåt avbrott i huvudprogrammet. Görs i SYSCON samt CPSR

KODMALL:

```
; Anpassar vektortabell. Lägger BRANCH-instruktion i avbrottsvektor
            ORG    0x00000018
            B      IRQ_handler

            ORG    ROM_START
            ; Initiering: stack, portar osv.
            ; Konfigurering av avbrottskälla i INTMOD, INTMSK + övriga
            inblandade register
            ; byt mode med CPSR. init IRQ-stack
            ; Tillåt avbrott med SYSCON och CPSR

; huvudprogram. I detta exemplet endast en evig loop
loop        B loop

; Avbrottshanterare
IRQ_handler ; spara arbetsregister
            ; avgör vilket avbrott i INTPND. Varje bit betyder ett
            specifikt avbrott
            ; anropa subrutin avbrott1 eller avbrott2
            ; kvittera avbrott i INTPND
            ; återställ arbetsregister
            SUBS PC, LR, #4          ; Återgång från avbrott.

; Kod för avbrott1
avbrott1    ; utför något
            MOV PC, LR  ; återgå

; Kod för avbrott2
avbrott2    ; utför något
            MOV PC, LR  ; återgå

            END
```

1. Konfigurera avbrottskälla 1 : TIMER

För att lösa uppgiften behövs två avbrottskällor. Ett timeravbrott för blinkning och ett externt avbrott för att läsa av knapp1. För att inte behöva skriva all kod utan att testa konfigurerar vi och testar timeravbrottet först.

Beskrivning

- Lysdioden ska från initialt blinka med frekvensen 1Hz. Denna ska senare kunna förändras genom att ställa in värde på LED0-LED2 och trycka på knapp 1 (se beskrivning nedan).

- Blinkfrekvensen ska kunna återställas till 1Hz genom att hålla nere knapp2 samtidigt som man trycker ner knapp1 (se beskrivning nedan)

- För att åter komma till 1Hz måste alltså systemet nollställas genom att hålla nere knapp2 samtidigt som man trycker ner knapp1.

Initiering av Timeravbrott

Vi väljer att använda 16-bitars TIMER0 i intervall-mode. Det innebär att man får timern generera avbrott vid jämna tidpunkter. För att konfigurera detta behövs 5 register:

För timer 0 gäller följande register:

- Konfigureras genom kontrollregister *rTCON0*
- Klockan kan skalas ner med ett prescalerregister *rTPRE0*
- Dataregister *rTDAT0*. Då timern når det värde som är skrivet i detta register genereras ett avbrott samt räknarvärdet nollställs
- Aktuellt räknarvärde i *rTCNT0*

För att tillåta TIMER0 avbrott slår man på det i avbrottsmaskregistret rINTMSK. Det avbrott vi vill tillåta är 'Timer0 match/capture interrupt'.

Tips:
Konfigurera ovanstående register på lämpligt sätt (se datablad). Tänk på storleken på registren då ni skriver alt. läser från dom! Sätt lämpligtvis pre-scalervärder så ni får jämt och bra värde till timern. Tex värdet 250. Ger en timerklocka på 25MHz/250 = 100kHz!

Då man konfigurerar maskregistret rINTEMSK kan det vara lämpligt att använda en bitmask med en etta satt för aktuellt avbrott. Det är därför det i koden finns två avbrottsmaskar definierade:

```
MASK_EINT0          EQU         0x00000001
MASK_TIMER0         EQU         0x00000100
```

Denna del är svår att testa utan resten av koden.

2. Konfigurering av IRQ-stack

ARM-processorn arbetar i olika mode. Beroende på vilket mode man är i har man lite olika rättigheter att förändra register. Man har även sk. bankade register, vilket innebär att varje mode har egen stack, statusregister och länkregister. Vi kommer att köra huvudprogrammet i supervisormode, men då avbrottet kommer växlar processorn automatiskt över till IRQ-mode. Det betyder att vi måste konfigurera stack för både supervisor och IRQ-mode. Supervisorstacken är redan konfigurerad i filen ni laddar ner eftersom man är i supervisormode från början.

Det som måste göras är att växla till IRQ-mode och initiera stacken. Växling av mode sker i statusregistret CPSR:s 5 lägsta bitar.

Skriv kod för att växla till IRQ-mode och sätta stacken! För beskrivning av bitar i statusregistret samt olika mode, se datablad. Tänk på att endast röra dom bitar som har med mode att göra.

För att läsa och skriva från/till statusregistret CPSR krävs två speciella instruktioner som kan komma till nytta:

```
MRS     R0,CPSR         ; Läser statusreg till R0
MSR     CPSR,R0         ; Skriver R0 till statusregister
```

3. Skriv avbrottshanterare.

Avbrottshanteraren eller handlern är ett kodparti som man ska hamna i då avbrottet sker. Den har i princip fyra uppgifter:

1. Avgöra vilket avbrott som skett
2. Hoppa till rätt avbrottskod
3. Kvittera rätt avbrott
4. Återgå till huvudprogram

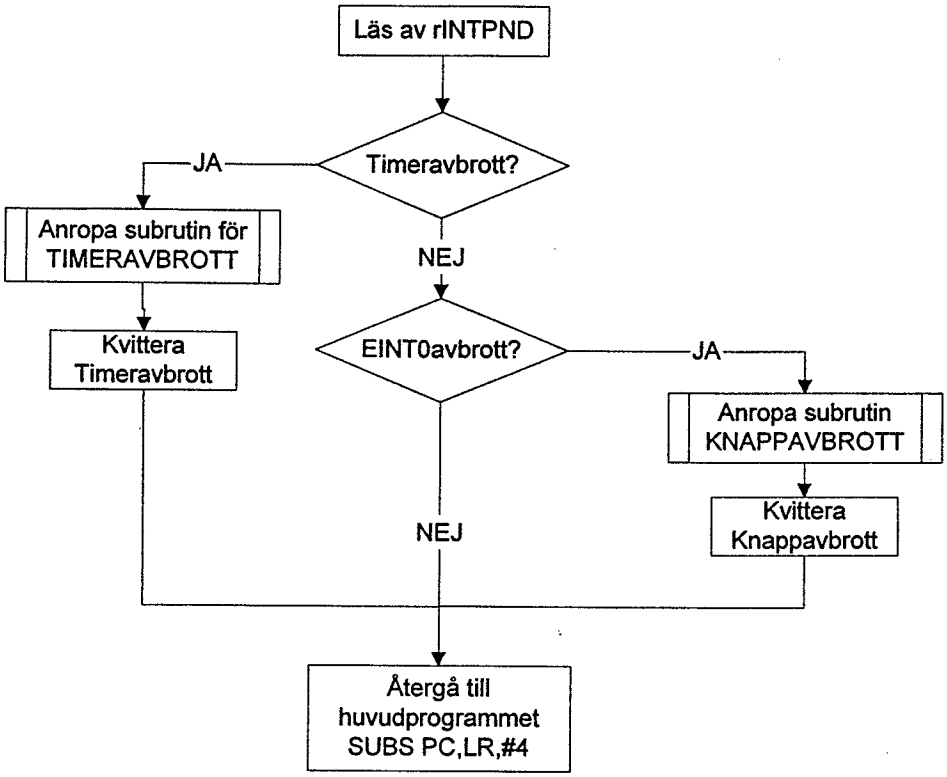
För att avgöra vilket avbrott som skett läser man av pendingregistret rINTPND. Det avbrott som genererat avbrottet sätter även en bit i registret. Lämpligt är att nyttja den avbrottsmask som är definierad för att läsa av registret. Eftersom vi kommer ha två avbrott får man testa ett efter ett. Beroende på vilket avbrott som skett anropar man olika subrutiner där man lagt den kod som ska utföras beroende på vilket avbrott, sk avbrottskod.

Efter avbrottskoden är exekverad kommer man ju återgå till avbrottshanteraren. Det sista man gör sen är att kvittera rätt avbrott (dvs nollställa den bit som var ettställd i pendingregistret. Även här kan man nyttja sin fördefinerade avbrotssmask.

Det sista som görs är att återgå till huvudprogrammet med specialinstruktionen:

```
SUBS    PC, LR, #4
```

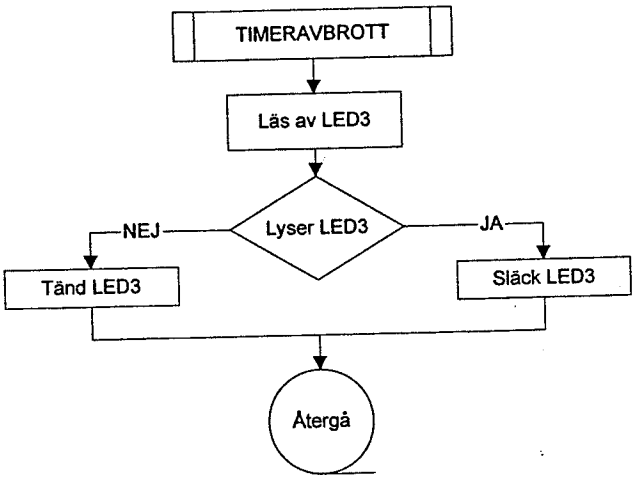
Födesschema för avbrottshandlern skulle kunna se ut så här.



Skriv kod för avbrottshandlern och kalla den `IRQ_handler`

4. Kod för timeravbrottet

Nu är det dags att skriva kod för vad som skall hända vid timeravbrottet. Det enda som vi vill ska hända är att lysdiod 3 antingen ska tändas och släckas beroende på tidigare tillstånd. Det blir en subrutin enligt nedan. Notera att man inte ska gå runt i någon form av loop för att uppnå blinkeffekten . Tänk på att koden kommer köras med jämna intervall på grund av timern!



Skriv subrutinen för timeravbrottet. Kalla den `TIMER`. Glöm inte att spara undan lämpliga register i subrutinen!

tips:
Istället för att kontrollera om LED3 lyser och slack/tända den kan man utföra operationen XOR mellan register med lysdiodvärde och en bitmask med en etta på den position man vill växla värde på. Se instruktionen EOR

5. Anpassa vektortabell

För att datorn ska kunna veta att vi vill den ska köra just vår avbrottskod vid avbrottet gäller det att lägga in ett hopp till vår kod på rätt sk. Avbrottsvektor. Då avbrottet kommer avslutar processorn vad den håller på med och hoppar till en specifik adress. I vårt fall den som gäller för IRQ-avbrott.

Ta ur datablad reda på vilken adress det blir och lägg där in en branschinstruktion till vår handler enligt nedan:

```
ORG   <specifik adress för IRQ-avbrott>
B     IRQ_handler
```

6. Tillåt avbrott i huvudprogrammet

Som sista del i initeringen ska man slå på avbrott. Detta görs i två register

a. Systemregister rSYSCON

b. Statusregister CPSR

Ta ur datablad reda på hur dessa register ska konfigureras och gör det! Tänk på att gå över i supervisormode innan du sätter registren! Då man arbetar mot systemregister som SYSCON är det extra viktigt att vara noga med att bara röra dom bitar man ska påverka. Oväntade händelser sker annars!

Testa timeravbrott!

Börja med att deltesta avbrottskoden, dvs subrutinen `TIMER`. Gör det genom att placera markören på den raden, högerklicka och välja **Set Next Statement**. Då sätts PC till den raden! Stega igenom din avbrottskod tills du är övertygad om att den fungerar!

Efter detta är det dags att prova avbrottet. Det går endast att göra i fullfart. Prova! Notera att huvudprogram som körs då man inte är i avbrottet är en evig loop

Vill du stega i din kod så sätt en brytpunkt på avbrottshandlern `IRQ_handler`. Då avbrottet väl kommit får du chansen att stega dig genom resten av koden.

Om det inte fungerar har du troligen gjort något av nedanstående fel:

- Fel i timerinitieringen.
- Glömt slå på avbrott
- Glömt initiera vektortabellen

Gå tillbaka och kontrollera koden noga!

Gå absolut inte vidare förrän du är säker på att avbrottet fungerar så långt! Fungerar allt så ska lysdioden blinka i 1Hz. Då är det dags att konfigurera knappavbrottet.

Konfigurera avbrottskälla 2: EINT0

Knapp1 är kopplad till P0:7. Tittar man i datablad över rPCON0 ser man att bit7 kan användas som external interrupt 0. För att slå på denna möjlighet måste följande register konfigueras:

- rEINTCON – Kontrollregister för External interrupt
- rEINTMOD – Ännu ett kontrollregister för externa interrupt. Ställer in på vilken flank/nivå som avbrottet ska reagera

Förutom detta måste man maska på avbrottet i rINTMSK. Använd lämpligtvis den fördefinierade avbrottsmasken för detta.

Konfigurera med hjälp av datablad EINT0-avbrottet så processorn reagerar på negativ flank (dvs när knappen trycks ner)!

Kod för EINT0-avbrottet

Då man kommit till till detta kodparti har knapp1 precis tryckts ner. Här ska två saker kunna hända

Om knapp 2 hålls inne då knapp 1 trycks ska nollställning ske, annars ska det inställda värdet på lysdioderna förändra blinkfrekvensen.

Förändring av blinkfrekvens

Då man trycker ner knapp 1 ska det värde som är inställt på lysdioderna LED0-LED2 förändra blinkfrekvensen.

- För varje inställt värde på lysdioderna LED0-LED2 ska olika frekvenser ställas . Detta är sammanställt i tabellen nedan:

LED2	LED1	LED0	Blinkfrekvens Hz]	rTDAT0
0	0	0	0.8	0x7A12
0	0	1	1	
0	1	0	2	
0	1	1	4	
1	0	0	8	
1	0	1	16	
1	1	0	32	
1	1	1	64	

- Det värde som är inställt på LED0-LED2 ska ligga kvar efter att blinkfrekvensen är inställd

Beräkna vilka värden som ska sättas på rTDAT0 enligt nedanstående metod. För in värdena i tabellen ovan!

Ex)
Processorklocka = 10MHz. Anta pre-scaler satt till 200. Ger en timerklocka på 50kHz. Anta vi vill ha en blinkfrekvens på 0.8Hz. Det betyder att avbrottet måste komma med dubbla frekvensen 1.6Hz eftersom att avbrottet både ska släcka och tända dioden!

1.6Hz => periodtid (1/f) = 0.625s.

Alltså ska timerna räkna upp till värde motsvarande 0.625s. Då timern får klockan 50kHz (periodtid 20us) måste den räkna upp till värdet

0.625s/20us = 31250 (hex 0x7A12). Det är detta värde som rTDAT ska innehålla!

För att ställa in de beräknade värdena läggs dom enklast i en tabell enligt nedanstående exempel:

```
; Anta R3 innehåller den position vi vill läsa från, tex 4
            LDR         R1,=TABELL        ; Adressen till tabellen

; läser ett 16-bitars tal i tabell från positionen R3. Talet hamnar i R2
; I vårt exempel hamnar talet 0x66 i R2

; Måste plocka jämna 16-bitarstal. Därför multiplicerar vi värdet i R3
; med 2 genom att vänsterskift 1 steg innan vi läser i tabellen.
            MOV         R3, R3, LSL #1
            LDRh        R2, [R1,R3]

; Tabellen läggs i slutet av koden
TABELL      DC16  0x12, 0x43, 0x33, 0x11, 0x66, 0x22, 0xC455, 0x34
```

Nollställning

Håller man inne knapp 2 samtidigt som knapp 1 trycks ska alla lysdioder släckas och LED3 ska åter blinka med 1 Hz.

Skriv kod för knappavbrottet och kalla den subrutinen `KNAPP`. Glöm inte att spara undan de register ni arbetar med.

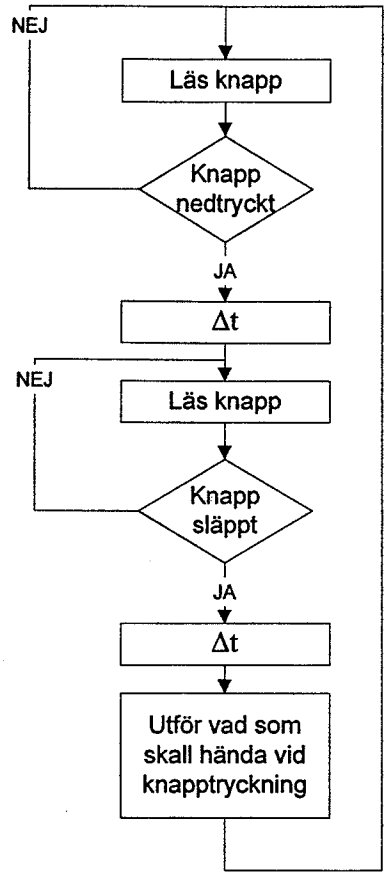
Testa knappavbrott!

Då man ska testa själva knappavbrottet kan det vara enklare att slå av timeravbrottet. Gör det genom att kommentera bort den raden som maskar på timeravbrottet i rINTMSK.

Testa knappavbrottet på samma sätt som timeravbrottet genom att sätta en brytpunkt på `IRQ_handler`. Kör fullfart och se om avbrott kommer vid knapptryckning. Stega genom avbrottskoden för knappavbrottet och övertyga dig om att den fungerar!

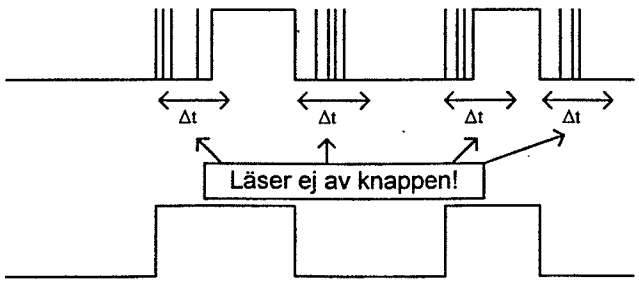
Glöm inte att slå på timeravbrottet igen när du testat klart

Flödesschema för knappregistrering blir som följer



Det gäller ju att fördröjningen ? t inte är för lång så man riskerar att missa riktiga knapptryckningar. 10-talet millisekunder är lämpligt för vanlig trycknapp. Man brukar inte kunna trycka så fort själv.

Ex)
I exemplet tänker vi oss man tryckt ner och släppt knappen 2 ggr. Hade man inte haft med fördröjningar skulle i vårt fall 16 knapptryckningar registrerats istället för två.



Genom att ha med fördröjningar ser datorn endast två pulser!
Precis vad vi ville!

Skriv kod för huvudprogrammet samt testa att det fungerar! Som delay kan man använda subrutinen `Delay_ms` som användes i förra laborationen!

Då du ska testa huvudprogrammet kan det vara lämpligt att inte ha på avbrotten. Slå av dom genom att kommentera bort delen där avbrott tillåts, dvs då man sätter CPSR och SYSCON.

Fungerar varje del är det dags att testa hela programmet! Slå åter på avbrotten (om du stängt av dom och kör hela programmet i full fart.

Om det inte fungerar:

- Stäng av avbrotten och deltesta varje kodparti. Övertyga dig om att varje del fungerar separat innan du låter dom arbeta tillsammans!
- Testa inte med avbrotten på om du inte är 100% säker att varje del fungerar som dom ska!
- Kontrollera stackhanteringen. Viktigt att du pushar (med STMFD) lika många register som du poppar (med LDMFD)!

Fungerar varje parti, men inte tillsammans:

- Det troligaste är att något gått fel i initieringen. Kontrollera detta noga igen!

Då programmet fungerar som det ska bör vi se till en sista sak, nämligen att skydda den arbete med den delade resurs vi använder.

Skydda huvudprogrammet

Då PORT0 blir gemensam resurs för både avbrottskod samt huvudprogram bör man skydda känslig kod i huvudprogrammet. Med känslig kod definieras kod som direkt arbetar med värde på PORT0, dvs från avläsning till tillbakaskrivning. Man bör se till att den koden görs så tajt (liten) som möjligt, samt slå av/på avbrott runt om den.

Problem kommer dock om timeravbrottet kommer mitt i där man förbjudit avbrottet. Det kommer då att missas, men vi tar den risken. Detta är dock alltid en avvägning som får göras från fall till fall. Avbrottet man missar servas så fort man slår på avbrotten igen, men blir dock något försenat!