Laboration 3 i Datorteknik- Avbrottshantering

Laborationens mål är att få förståelse för avbrottshantering.

- Ladda hem filen Lab3_DO2005.zip och packa upp på lämpligt ställe på ditt konto. Där i ligger färdiga projektfiler och källkod för uppgift l. Öppna projektet LAB3_1.eww. Där finns en kodmall liknande den som beskrivs nedan.
- Bilagor till labben är om inget annat anges *SAM3U Series Complete* - http://www.atmel.com/Images/doc6430.pdf och *Getting Started with SAM3U Microcontrollers* - http://www.atmel.com/Images/doc11020.pdf

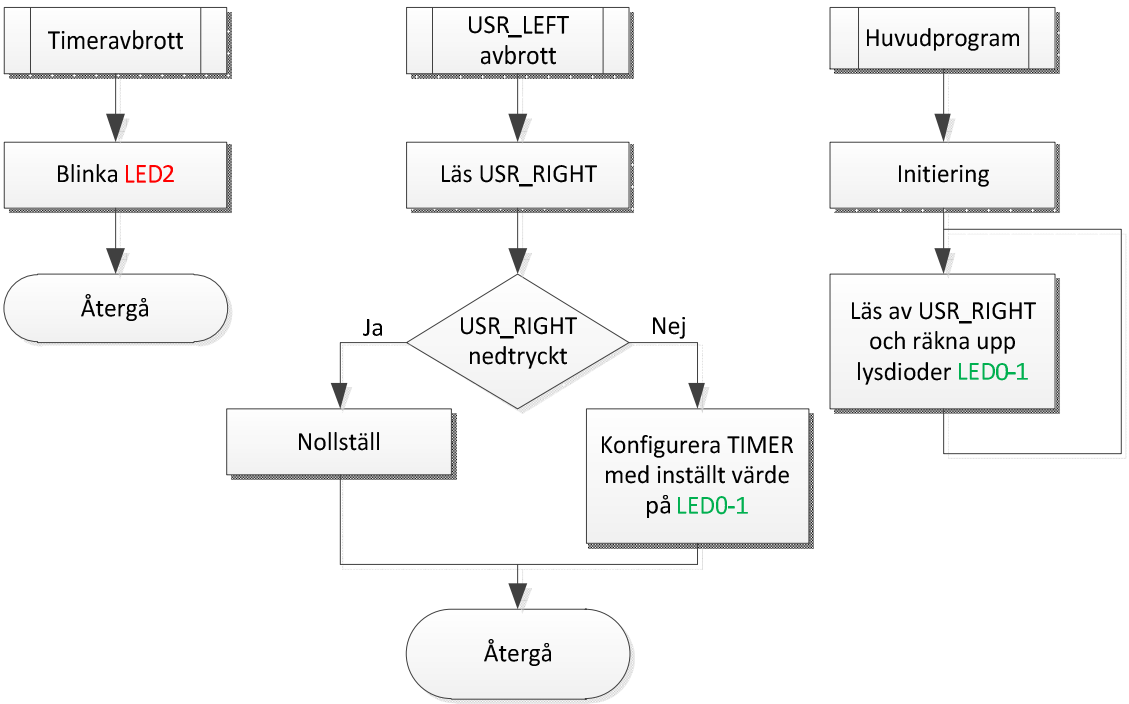
Målet med laborationen är att konstruera ett system enligt nedanstående.

Grundfunktion hos programmet

Laborationsplattformen har tre lysdioder och två knappar. De kommer att refereras till som LED2-LED0 samt USR_LEFT och USR_RIGHT.

LED2	LED1	LED0	USR_LEFT	USR_RIGHT

- LED2 ska blinka med en viss frekvens.
- Frekvensen ska kunna ställas med USR_RIGHT. Då USR_RIGHT trycks skall LED0-LED1 räkna upp binärt. Genom att trycka på USR_LEFT ska det binära tal som nu är inställt på LED0-LED1 förändra frekvensen på den blinkande LED2. Man ställer alltså in med USR_RIGHT och bekräftar med USR_LEFT
- Genom att hålla inne USR_RIGHT samtidigt som man trycker USR_LEFT ska alla lampor nollställas samt systemet nollställas, d.v.s. det ska vara som från början. En form av reset-funktion.
- Programmet ska använda timeravbrott samt avbrott från USR_LEFT och vara uppbyggt enligt nedanstående princip



Arbetsgång

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

1. Konfigurera avbrottskällor.
2. Skriv avbrottskod för de avbrott man vill använda.
3. Anpassa vektortabellen till avbrottshanteraren som tar hand om avbrottet.
4. Tillåta avbrott i huvudprogrammet.

Kodmall (en del av koden visas här, hela koden bifogas):

```
      SECTION .intvec : CODE (2)
      THUMB
      DATA
__vector_table
      DCD   0x20008000            ; initiering av Stackpekare
      DCD   __iar_program_start   ; Reset vektor
      DCD   NMI_Handler           ; Interrupt handler
      DCD   HardFault_Handler
      DCD   MemManage_Handler
      DCD   BusFault_Handler
      DCD   UsageFault_Handler
      DCD   0
      DCD   0
      DCD   0
      DCD   0
      DCD   SVC_Handler
      DCD   DebugMon_Handler
      DCD   0
      DCD   PendSV_Handler
      DCD   SysTick_Handler
; Configurable interrupts ----------------------------
; Skriv vektortabell för externa avbrott här

      SECTION .text : CODE (2)
      THUMB
; Program start-------------- -------------------------
__iar_program_start
; Huvudprogram ----------------------------------------
STOP  B       STOP

; Interrupt handler -----------------------------------
NMI_Handler
      B NMI_Handler
HardFault_Handler
      B HardFault_Handler
MemManage_Handler
      B MemManage_Handler
BusFault_Handler
      B BusFault_Handler
UsageFault_Handler
      B UsageFault_Handler
SVC_Handler
      B SVC_Handler
DebugMon_Handler
      B DebugMon_Handler
PendSV_Handler
      B PendSV_Handler
SysTick_Handler
; Skriv din kod för Timeravbrott här
      BX    LR                    ; Return interrupt
      B SysTick_Handler
; ----------------------------------------------------------
      END
```

1. Konfigurera avbrottskälla 1: SysTick

För att lösa uppgiften behövs två avbrottskällor. Ett avbrott för blinkning och ett externt avbrott för att läsa av USR_LEFT. 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-LED1 och trycka på USR_RIGHT (se beskrivning nedan).
- Blinkfrekvensen ska kunna återställas till 1Hz genom att hålla nere USR_RIGHT samtidigt som man trycker ner USR_LEFT (se beskrivning nedan)
- För att åter komma till 1Hz måste alltså systemet nollställas genom att hålla nere USR_RIGHT samtidigt som man trycker ner USR_LEFT.

Initiering av SysTick

Vi väljer att använda SysTick avbrott. Den har en 24-bitas räknare som kan konfigureras så att den genererar ett avbrott vid jämna intervall beroende på startvärde på räknaren. För att konfigurera detta behövs 3 register (se vidare Kap 13:21).

För att konfigurera SysTick Timer görs följande:

- Slå av SYSTICK genom att skriva 0 i ”SysTick Control and Status Register”
- Skriv startvärde för räknaren i ”SysTick Reload Value Register”
- Skriv 0 till ” SysTick Current Value Register” för att nollställa aktuellt värde.
- Skriv till ”SysTick Control and Status Register” för att starta SysTick timer.

I kodmallen som följer med är redan initiering av vektortabell gjord och en mycket enkel avbrottshanterare är implementerad, en loop, se `SysTick_Handler`.

Beräkning av tid mellan avbrott

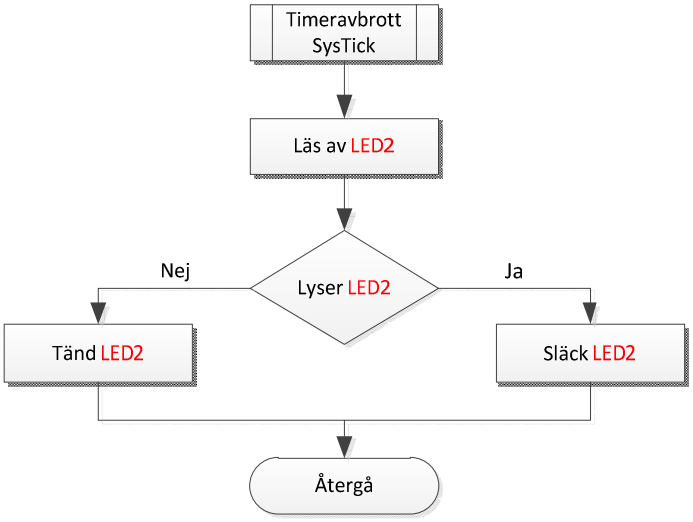
Det värde som skall läggas som startvärde i ”SysTick Reload Value Register” beror på vilken period som man vill att avbrottet skall köra med. Räknaren räknar ned till noll och då genereras ett avbrott och räknaren börjar om igen. Hastigheten som räknaren räknar ner beror på MCK (Master Clock) och vald klockkälla i ”SysTick Control and Status Register”. MCK beror på den frekvens som processorn kör med (Default frekvensen).

Ex. Är MCK = 10.5 MHz och man vill ha en periodtid på 1 ms blir det: $10.5*10^6$ *0.001 = 10500 eller 0x2904.

Se vidare Kap 13:21, Kap 27:1-4, Kap 10:5-6 i *SAM3U Series Complete.*

Kod för timeravbrottet

Det enda som vi vill ska hända är att LED2 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!



Testa timeravbrott!

Börja med att testa avbrottskoden. Gör det genom att placera brytpunkt på första raden i avbrottskoden. Stega igenom din avbrottskod tills du är övertygad om att den fungerar!

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

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.

Frågor

1. Vad händer med stacken vid avbrott?…….
2. Vad är det som sparas undan på stacken vid avbrott?……
3. Vad betyder det som står i LR-registret (jämför med subrutin)? …….....
4. Vilken instruktion är det som gör att avbrottet avslutas, d.v.s. man återgår till huvudprogrammet?.....
5. Var lagras återhoppsadressen?.....

2. Konfigurera avbrottskälla 2: IRQ-avbrott PORTA

USR_LEFT är kopplad till bit18 på PORTA. I lab 1 så slog vi av möjligheten till avbrott genom att ettställa PIOA_IDR registret. För att få avbrott att utföras då vi aktiverar USR_LEFT behöver följande göras:

1. Anpassa vektortabellen för externt avbrott på PORTA.
2. Lägga till en avbrottshandler/avbrottsrutin.
3. Initiera PORTA för avbrott på bit18 (USR_LEFT).
4. Konfigurera NVIC – registret för externt avbrott på PORTA.
5. Skriva avbrottskod (det vi vill ska utföras vid avbrott).

1. Anpassa vektortabell

I kodmallen finns bara vektortabell för systemavbrott (1-15). Denna behöver nu utökas med de externa avbrotten. Det som skall göras är att man skall skriva följande:

```
DCD  PIOA_IrqHandler        ; Detta är en label som pekar på avbrottsrutinen
```

Men man behöver skriva denna på ett speciellt ställe i avbrottstabellen. Eftersom vi bara kommer att använda oss av ett externt avbrott kan vi skriva DCD 0 på övriga ställen. Inte heller behöver tabell skrivas för avbrott som kommer efter IRQ-avbrott PORTA i tabellen. Hur avbrottstabellen skall se ut framgår av hur tabellen ser ut för systemavbrotten samt Kapitel 13.5 och då speciellt Figur 13.3. Vidare kan man läsa i *Getting Started with SAM3U Microcontrollers* sid 6 för vägledning.

2. Lägga till avbrottshandler

Efter SysTick_Handler i programmet så lägger man kod för IRQ-avbrottet.

```
PIOA_IrqHandler
; Här skriver du din avbrottskod senare
     B    PIOA_IrqHandler
```

3. Initiera PortA för avbrott.

Skriv kod i som initierar avbrott för PORTA bit18 (USR_LEFT). PIO_IER (Interrupt Enable Register) och PIO_IDR (Interrupt Disable Register) slår på respektive av om avbrott initieras vid tillståndsförändring/flank (edge) eller ett viss nivå (level) på en inport. Nuvarande värde kan läsas ifrån PIO_IMR (Interrupt Mask Register). [=0 inget avbrott tillåts, =1 kan generera avbrott]. Om avbrott tillåts så kan man styra vad som ska trigga avbrottet styrs via 3 Huvudregister till, se sid 517 i *SAM3U Series Complete*.

Konfigurera med hjälp av datablad avbrottet så processorn reagerar på negativ flank (d.v.s. när knappen trycks ner (Falling Edge))!

När en ingång upptäcker en avbrottshändelse (flank eller nivå) på en I/O pinne så sätts motsvarande bit i PIO_ISR (Interrupt Status Register). Om sedan den motsvarande biten i PIO_IMR är satt så generas ett processoravbrott. Alla 32 avbrottssignalerna ifrån en port ORas samman så att en enda avbrottssignal går till NVIC (Nested Vector Interrupt Controller). Men vilken pinne som gav avbrott kan ju avläsas i PIO_ISR.

4. Konfigurera NVIC register för avbrott (i huvudprogrammet)

Externt avbrott slås på genom att ettställa motsvarande bit i SETENA ”Interrrupt set enable register” och slås av genom CLRENA ”Interrrupt clear enable register” i NVIC. Om ett avbrott väntar på att få köras så ettställs motsvarande bit i SETPEND ”Interrupt Set-pending Registers” (Läsning av register. Man kan också läsa från CLRPEND). Om ett avbrott är aktiverat syns detta syns detta genom i ACTIVE ” Interrupt Active Bit Registers”.

Innan man slår på ett avbrott är det en god sed att nollställa alla ”pending” avbrott. Detta görs genom att skriva en etta till CLRPEND ”Interrupt Clear Pending Register”.

Läs vidare i Kap 13.19 (Tabell 13-27 och 13-28) samt Kap 13.5 (Tabell 13.3).

5. Avbrottskod

Då man kommit till detta kodparti har USR_LEFT precis tryckts ner. Här ska två saker kunna hända. Om USR_RIGHT hålls inne då USR_LEFT trycks ska nollställning ske, annars ska det inställda värdet på lysdioderna förändra blinkfrekvensen.

Förändring av blinkfrekvens

Då man trycker ner USR_LEFT 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:

LED1	LED0	Blinkfrekvens	VALUE
0	0	0.5	
0	1	1	
1	0	2	
1	1	4	

OBS! Anta vi vill ha en blinkfrekvens på 0.5 Hz. Det betyder att avbrottet måste komma med dubbla frekvensen 0.25 Hz eftersom att avbrottet både ska släcka och tända dioden!

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

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

; läser ett 32-bitars tal i tabell från positionen R3. Talet hamnar i R2
; I vårt exempel hamnar talet 0x66 i R2

; Måste plocka jämna 32-bitarstal. Därför multiplicerar vi värdet i R3
; med 4 genom att vänsterskift 2 steg innan vi läser i tabellen.
            MOV   R3, R3, LSL #2
            LDR   R2, [Rl,R3]

; Tabellen läggs i slutet av koden
            DATA
TABELL      DCD   0xl2, 0x43, 0x33, 0xll, 0x66, 0x22, 0xC455, 0x34
```

Nollställning

Håller man inne USR_RIGHT samtidigt som USR_LEFT trycks ska alla lysdioder släckas och LED2 ska åter blinka med 0.5 Hz.

Tips! Huvudprogrammet som vi skall skriva senare hanterar lysdioderna. Det kan vara problematiskt att släcka dessa i avbrottrutinen eftersom vi inte vet var i huvudkoden som avbrottet sker, exempelvis precis innan man tänder en lysdiod. Ett sätt att lösa detta är att man sätter en flagga (bit i något register) som sedan huvudprogrammet tolkar som reset. Annat alternativ är att man förhindrar att knappavbrott sker då man ändrar lysdioderna i huvudprogrammet. Mer om detta senare.

Tips! Det kan också vara idé att man väntar tills båda knapparna är släppta innan man går vidare i koden, d.v.s. man ligger och väntar tills man släppt knappen.

Testa knappavbrott!

Då man ska testa själva knappavbrottet kan det vara enklare att slå av SysTick avbrottet. Gör det genom att kommentera bort den raden som slår på avbrottet.

Testa knappavbrottet på samma sätt som timeravbrottet genom att sätta en brytpunkt på `PIOA_IrqHandler`. 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.

Frågor

1. Vad händer om man får ett av knappavbrott då man kör koden? Testa genom att trycka på USR_LEFT när du stegar igenom avbrottskoden. Kolla CLRPEND0 registret med *View Memory*.....
2. Hur kan man undvika att man direkt hoppar till avbrottet igen om man får ett avbrott under avbrottshanteringen?.....

3. Huvudprogram

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.

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 dem 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 sparar undan register som du använder i dina avbrott respektive subrutiner och de inte sparas undan på annat sätt (gäller avbrott)!

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, som vi berört tidigare, nämligen hur vi skall hantera när vi arbetar med delade resurser, d.v.s. tända LED och läsa av knappar.

Delade resurser

Då lysdioder och knappar blir gemensam resurs för både avbrottskod samt huvudprogram bör man skydda känslig kod i huvudprogrammet. Med känslig kod definieras kod som direkt arbetar med manipulera lysdioder och kappar, d.v.s. allt från avläsning till tillbakaskrivning. Man bör om möjligt 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 avbrott. 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!

Se till att ditt program inte hamnar i en situation där delade resurser kan utnyttjas av både huvud program och avbrott samtidigt. Ex. nollställning av lysdioder i avbrott, men programmet hoppar tillbaka i huvudprogrammet så tänds de igen.