Laboration Datorteknik TSIU02/TSEA28
3. Digitalur

Michael Josefsson, Kent Palmkvist

Version 3.3, Februari 2022

Innehåll

1. Inledning **5**

2. I/O enheter **7**
2.1. LED-displaymodul 7
2.1.1. Hårdvarumodul 7
2.1.2. Distanslägesversion 8
2.1.3. Multiplexning 8
2.1.4. Övrigt 9
2.2. Tidbas 10

3. Laborationsuppgifter **11**
3.1. Krav på avbrottsrutiner och huvudprogram 11
3.2. Multiplexning av displayen (avbrottsrutinen `MUX`) 11
3.3. Tidräkning (avbrottsrutinen `BCD`) 14

4. Sammanfattning **19**

A. Schema på sjusegmentsmodulen **21**

1. Inledning

I den här laborationen skall du konstruera ett digitalur som visar minuter och sekunder. Till din hjälp har du flera labmoduler som kommer att underlätta konstruktionen. Du kommer att använda en s k 7-segmentsdisplay för att visa siffrorna och PIA:n för att kommunicera mellan den och ditt program. Tiden skall hållas exakt med hjälp av en tidbasmodul som ger en puls varje sekund. Modulen ger också pulser med frekvensen 10, 100 och 1000 Hz. De högre frekvenserna skall du använda för att *multiplexa* programmets utsignaler till displayen.

Observera.

Laborationshandledningen beskriver inte någon detaljerad rekommenderad arbetsgång!

Du måste själv tänka ut lämpliga tester för att avgöra om programrutinerna fungerar. Gå inte vidare utan att först vara *helt övertygad* om att dina programrutiner fungerar som avsett. Det är mycket lättare att rätta till ett fel tidigt i konstruktionen.

Även om arbetsgången inte är bestämd finns det vissa förberedelseuppgifter. Förberedelseuppgifterna är markerade med **fetstil** i handledningen. Dessa skall vara klara innan du påbörjar laborationen och skall i allmänhet redovisas för laborationsassisten under laborationens gång. Med "**Uppgift**" markeras nödvändiga programmerings- och uppkopplingsuppgifter som behövs för laborationen. Dessa är ett minimum av vad som behövs. Du kan behöva göra andra kopplingar och/eller programmeringar för att testa att allt fungerar som det skall.

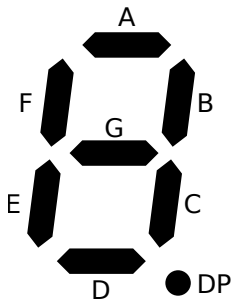
2. I/O enheter

Till laborationen hör en del speciell hårdvara i form av labmoduler som beskrivs nedan. Utöver de beskrivna modulerna kommer du att använda den normala laborationshårdvaran med Darma-systemet.

2.1. LED-displaymodul

För att visa tiden används en modul med fyra 7-segmentsdisplayer. De enskilda siffrorna består av sju[1] lysdioder kopplade enligt nedan:

2.1.1. Hårdvarumodul



En sju-segmentssiffra. De olika segmenten är namngiva `A`–`G` och `DP` för decimalpunkt. Modulen består av fyra sådana siffror. Decimalpunkten behöver inte användas i laborationen.

I bilaga A kan vi studera hur modulen är konstruerad. I schemat ser vi att segmentens olika dioder är kopplade med s k *gemensam anod* med vilket menas att varje siffra i modulen har gemensam spänningsmatning. För att tända ett segment måste man förse siffrans anod med plus-spänning samt jorda det segment man vill tända. Modulens konstruktion gör den lätt att använda i och med att den *utifrån* använder *positiv logik*, dvs för att tända ett segment skall detta förses med logisk etta (5 V). Detta genomförs med transistorerna T5–T12, vilka leder ström då deras bas får en spänning på sig.

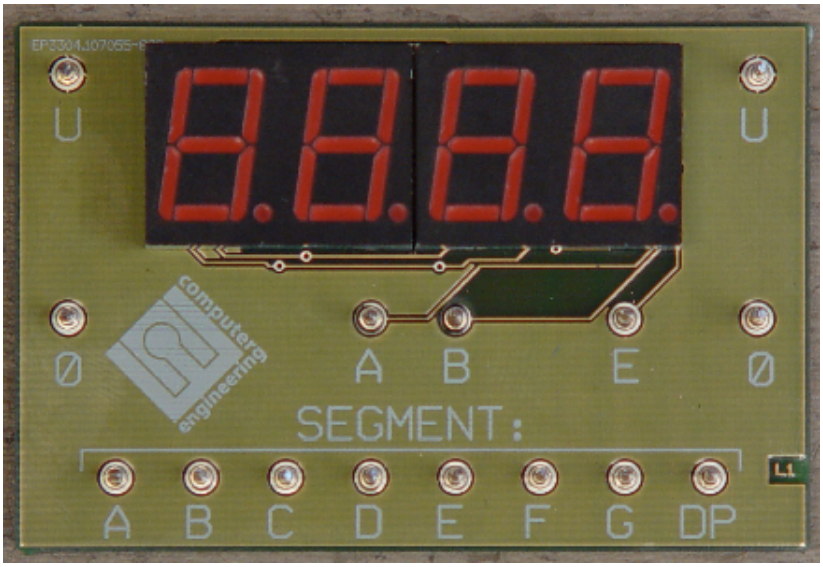
Modulen består av två par sådana sjusegmentsdisplayer monterade i en komponent. För att ändå kunna tända och släcka siffrorna individuellt är de olika siffrornas gemensamma

[1] Egentligen åtta, eftersom decimalpunkten också kräver en lysdiod. Men det bortser man från i det allmänna språkbruket och kallar det sjusegmentsdisplayer i alla fall.

anoder utdragna till en avkodare (IC1). Respektive anod är även kopplad via en transistor (T1–T4) då avkodaren inte kan leverera de strömmar som behövs för att segmenten skall lysa bra.

För att skriva en siffra kan man behöva tända flera dioder samtidigt. Siffran ”7” kan till exempel fås genom att spänningssätta segmenten A, B och C eller A, B, C och F.

Utan logisk 1 på signalen E (Enable) kommer displayen att vara släckt hela tiden. Denna insignal ansluts inte via någon pinne på Darmas portar, utan kopplas till +5 V permanent.



LED-displaymodulen. Med segmentingångarna `A`–`G` och `DP` väljs vilket eller vilka segment som skall vara tända. De under siffrorna belägna ingångarna `A` och `B` används för att välja vilken siffra som skall tändas. `A` är minst signifikant bit och siffrorna är numrerade i ordningen {3, 2, 1, 0}. Ingången `E` måste vara hög för att displayen överhuvudtaget skall lysa.

2.1.2. Distanslägesversion

I det fall laborationen körs på distans används istället programmet tsea28lab3. Detta program simulerar effekten av en LED-displaymodul som är ansluten till port B bit 0-7 samt port F bit 1 och bit 0. Effekten av att snabbt byta mellan olika siffror är dock inte lika bra som för den fysiska modulen, men tillräcklig för att det ska synas om lösningen på laborationsuppgiften är korrekt.

2.1.3. Multiplexning

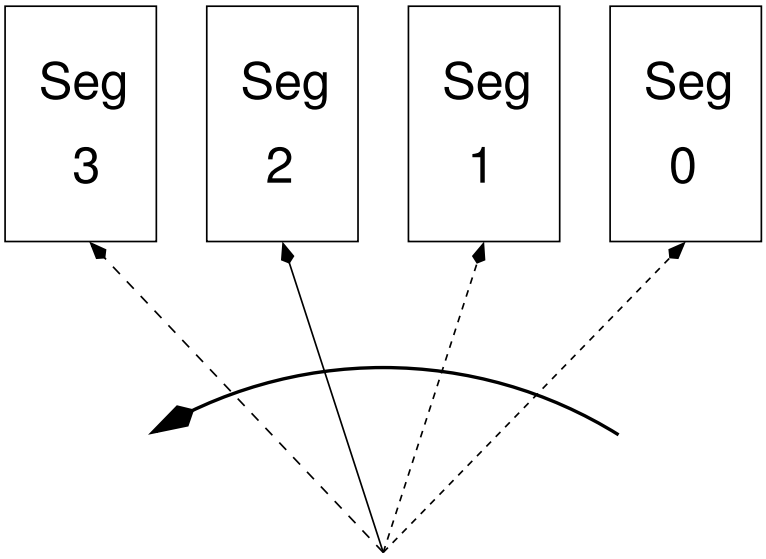
Modulens grundkonstruktion tillåter oss att visa en siffra i taget. Med *multiplexning* kan vi dock visa siffror i samtliga positioner.

Multiplexning går till så att vi visar en siffra en stund, sedan växlar vi snabbt över till nästa siffra och visar denna en stund, och så vidare, tills alla siffror visats, varefter vi börjar om från början igen. Gör vi detta tillräckligt snabbt kommer ögat att uppfatta det som att alla positioner lyser hela tiden. Eftersom siffran nu i verkligheten blinkar, blir det något svagare ljus än om man tillät den att lysa kontinuerligt, men det är i modulen redan kompenserat genom att man ökat strömmen något.

> I programmet är det alltså under tiden *mellan* två avbrott som en siffra ska lysa. Mellan *nästa* avbrott ska nästa siffra lysa osv

Praktiskt utförs multiplexningen genom att

1. Peka ut rätt önskad position med signalerna `A` och `B` på displaymodulen.
2. Lägga på önskat bitmönster på segmentingångarna `A`–`G` och ev `DP`.
3. Vänta, så att siffran får lysa en stund. Typiskt ett tiotal millisekunder.
4. Gå till 1 och gå igenom sekvensen igen med nästa position.



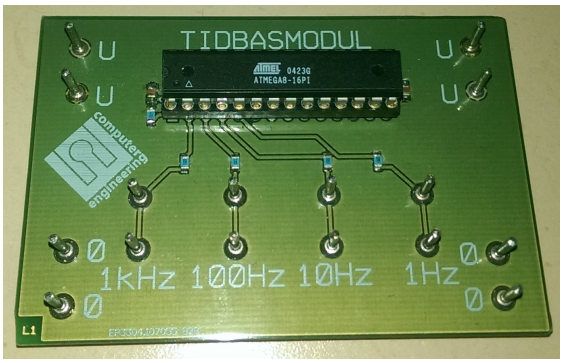
Med multiplexning visas varje siffra bara så länge som behövs för att bilden inte skall flimra.

2.1.4. Övrigt

Om för stor ström tillåts gå genom lysdioden bränns denna upp varför någon form av *strömbegränsning* måste införas. På modulen är detta genomfört på enklaste sätt med motstånden R9–R16. Spänningsfallet över motståndet sker enligt Ohms lag så att $U = R \cdot I$. Med ökande ström kommer alltså spänningsfallet över motståndet att bli större varför spänningen över dioderna blir mindre. Men då varje segment har sitt eget

motstånd kommer strömmen genom detta vara beroende enbart av det tända segmentet, och då detta segment alltid drar samma ström blir ljusstyrkan konstant. Hade strömbegränsningsmotstånden suttit i anoden skulle strömmen däremot, och därmed ljusstyrkan, varierat.[2]

2.2. Tidbas



Tidbasen ger flera utsignaler i form av pulser. Pulserna återkommer med frekvensen 1, 10, 100 och 1000 Hz. Frekvensen 1 Hz är lämplig att använda för att ticka fram sekunderna i digitaluret. Modulen behöver inga inställningar. När matningsspänningen ansluts börjar den automatiskt att ge pulser.

[2] Även de transistorer som sitter i anodtilledningarna har ett något varierande spänningsfall över sig men det är dels litet (några tiondels volt), dels i stort sett oberoende av kollektorströmmen (några millivolt hit eller dit spelar ingen roll för vår tillämpning).

3. Laborationsuppgifter

För att hinna göra laborationen på utsatt tid krävs att du läst igenom — och förstått! — *hela* laborationshäftet. Det finns förberedelseuppgifter insprängda som också måste vara avklarade *innan* laborationen och som skall redovisas för laborationsassistenten.

För denna laboration är förberedelseuppgifterna huvudsakligen:

- Bestämma siffrornas utseende.
- Skriva avbrottsrutinen `MUX`
- Skriva avbrottsrutinen `BCD`

Samtliga förberedelseuppgifter är tydligt angivna i med rubriken "**Förberedelseuppgift**".

3.1. Krav på avbrottsrutiner och huvudprogram

Det är viktigt att en avbrottsrutin är skriven på ett sådant sätt att den ej ändrar på huvudprogrammets tillstånd. Det vill säga, efter att en avbrottsrutinerna i denna laboration har körts måste alla processorns register ha samma värde som innan avbrottsrutinen började köras.

Avbrottsrutinerna kan inte heller förutsätta speciella värden i registren när avbrott sker. Detta för att framtida utveckling av huvudprogrammet ska tvingas reservera vissa register.

Utöver detta är det också ofta viktigt att ett system skrivs på ett sådant sätt att ett strömsparläge kan användas. I denna laboration ska huvudprogrammet vara skriven på ett sådant sätt att strömsparläge aktiveras så snart inget avbrott pågår. (Se dokumentationen för instruktionen `WFI` för mer information om hur strömsparläge i Cortex-M-baserade system kan aktiveras.)

Avbrottsrutinerna behöver inte vara namngiva BCD och MUX (det går bra att behålla namnen intgpiod och intpgioe), men indikera då vilken som är MUX respektive BCD mha kommentarer i början av avbrottsrutinen.

Lagring av tid ska ske i BCD-format, med en byte per siffra. Uppdatering av siffror ska ske av Avbrottsrutinen MUX ska bara använda BCD-informationen om tid för att bestämma utseendet på displayen.

3.2. Multiplexning av displayen (avbrottsrutinen `MUX`)

Vi antar att tiden återfinns BCD-kodad i minnesceller med början i adress `0x20001000` och med en BCD-siffra per byte enligt följande exempel.

Exempel

BCD-kodning av tid. Tiden 04:53 (4 minuter, 53 sekunder) anges i minnet av ett 32-bitars tal enligt $\boxed{00}\boxed{04}\boxed{05}\boxed{03}$. Tiden finns alltså inrymd i fyra bytes med en byte per siffra. Minnesbyten `0x20001000` innehåller här ”3”, `0x20001001` ”5” osv.

■

Programmet som skall visa siffror i dessa positioner på displayen är en avbrottsrutin som aktiveras av tidbasmodulen.

Förberedelseuppgift 1 För att visa en siffra måste sjusegmentsmodulens A–F-ingångar förses med lämplig information så att rätt lysdioder tänds. Det bitmönster (8 bitar) som skall skickas ut vid respektive siffra återfinns i en tabell i programmet. Men tabellen är inte komplett. Fyll i tabellen nedan med resterande bitmönster. Notera att bitmönstren måste anges i hexadecimal form. Du får själv välja utseende på de visade siffrorna.

```
SJUSEGTAB    .byte 0x3F    ; '0'
             .byte         ; '1'
             .byte         ; '2'
             .byte         ; '3'
             .byte         ; '4'
             .byte         ; '5'
             .byte         ; '6'
             .byte         ; '7'
             .byte         ; '8'
             .byte         ; '9'
```

Tips. Är du osäker på hur tabellen fungerar översätt då det givna värdet för siffran ”0”, `0x3F`, till ett binärt värde och matcha dessa mot de segment displayen behöver tända för att visa siffran `0`.

Förberedelseuppgift 2 Skriv en avbrottsubrutin, `MUX`, som visar rätt siffra på rätt position!

`MUX:`

Tips! För att säkerställa att rätt siffra visas på rätt position på displayen måste dessa vara synkroniserade. Ett sätt kan vara att använda en separat modulo-4-räknare (som ger sekvensen 0→1→2→3→0...) och addera BCD-siffrornas basadress (`0x20001000`) till denna. Då kan modulo-4-sekvensen styra vilken siffra på displayen som skall tändas och innehållet i `0x20001000`–`0x20001003` peka ut det data som skall användas.

Enklare är att bara använda en räknare, för adresserna. Om denna adressräknare går igenom sekvensen `0x20001000`→`0x20001001`→`0x20001002`→ `0x20001003` →`0x20001000`..., kan adressens två minst signifikanta bitar användas för att styra displayens multiplexingångar (`00`, `01`, `10`, `11`, `00`,...). Smart va!

Exempel

Siffran '2' skrivs ut. Displayen skall då förses med bitmönstret från denna rad i `SJUSEGTAB`. Rätt byte fås genom att addera tabellens startadress med den önskade siffran. Slutlig adress = tabellstart (`SJUSEGTAB`) + index (0–9).

■

```
AND    r1,r0,#0x0F    ; Maska ut siffran (fyra bitar),
```

```
                        ; antar att den finns i R0 här
ADR     r2,SJUSEG       ; Tabellstart till R2
ADD     r2,r2,r1        ; Peka ut rätt byte
LDRB    r1,[r2]         ; Hämta bitmönstret
```

I ARM kan man också använda den mer komplexa adresseringsmoden *Register Indirect with register indexing* med instruktionen `LDRB r1,[r2,r1]`.

Vad händer om programmet råkar peka utanför tabellen? Gör det nåt? Hur kan detta hanteras?

Uppgift 1 Programmera, koppla upp hårdvara och testa `MUX`!

GPIO port B och F initieras i initGPIOB och initGPIOF (i lab2.asm mall) och sätts där som utgångar. Glöm inte att även uppdatera tm4c123gh6pm_startup_ccs.c så avbrottsrutinerna hittas. Behåll namnen från lab2.asm men notera med en kommentar vilken som är MUX och vilken som är BCD i filen.

Än så länge finns ingen vettig tid att visa. Mata in en lämplig (fast) tid i `0x20001000`–`0x20001003` exempelvis genom att använda memory browser. Sätt värdena till 0x01, 0x02, 0x04, 0x08. Vilken tid motsvarar detta?

3.3. Tidräkning (avbrottsrutinen `BCD`)

Hittills har vi ett program som kan skriva ut siffror på en display. Vi saknar fortfarande kopplingen till sekunder och minuter. Vi måste tillverka en rutin som uppdaterar tiden i minnet (dvs minnescellerna `0x20001000`–`0x20001003`) så att de *alltid* är giltiga siffor. Denna rutin utförs som en avbrottsrutin som ska anropas varje sekund!

En lämplig talkodning för tiden är BCD, *Binary Coded Decimal*. Vi låter de fyra lägsta bitarna i varje byte innehålla information om vilken *decimal* siffra byten innehåller. Fyra bitar kan rymma siffrorna 0–16 men med BCD-kodning tillåter vi enbart 0–9. Efter detta måste den mer signifikanta siffran räknas upp.

När vi handhar tid kan vissa siffror bara anta värdet 0–5 innan den mer signifikanta siffran skall räknas upp. Vår BCD-räknare för tid skall alltså räkna upp segmenten enligt följande:

Tid	3	2	1	0
00:00	0	0	0	0
|	|	|	|	|
00:09	0	0	0	9
00:10	0	0	1	0
00:11	0	0	1	1
|	|	|	|	|
00:19	0	0	1	9
00:20	0	0	2	0
00:21	0	0	2	1
|	|	|	|	|
00:59	0	0	5	9
01:00	0	1	0	0
01:01	0	1	0	1
|	|	|	|	|
59:59	5	9	5	9
00:00	0	0	0	0
00:01	0	0	0	1

Minuträknaren ska räkna upp till 59. Därefter ska den börja om på noll.

Förberedelseuppgift 3 Skriv en subrutin som, för varje anrop, räknar upp de fyra minnesscellerna som ovan.

```
BCD:
```

Uppgift 2 Sätt ihop programmets delar till en fungerande klocka och anslut sekundpuls till `BCD`-rutinen.

Om allt stämmer skall programmet nu vid avbrott läsa av minnescellerna som innehåller tiden och uppdatera dessa. Vi har två program som körs helt oberoende av varandra. `BCD`-avbrottsrutinen körs på anmodan av den yttre sekundpulsen medan `MUX`-avbrottsrutinen är helt ovetande om att detta sker! Huvudprogrammet exekveras så fort processorn kan men utför inga ”nyttiga” instruktioner, hela klockan styrs helt och hållet av de två avbrottsrutinerna.

Beroende på hur du löst uppgiften kan man ibland skymta att även andra segment lyser upp. Detta försämrar kontrasten hos siffrorna och är inte önskvärt. Vidtag en lämplig åtgärd mot detta? (Ofta behöver bara en assemblerrad läggas till eller möjligen flyttas).

Extrauppgift I mån av tid och intresse finns det givetvis stora möjligheter att lägga till funktionalitet till det program du har konstruerat i denna uppgift. I detta avsnitt finns det tips på några möjliga självstudieuppgifter du kan göra, antingen i samband med laborationen eller senare i samband med att du förbereder dig för tentan.

- Alarmfunktionalitet: När klockan har nått en viss tidpunkt ska detta visas för användaren (exempelvis genom att siffrorna börjar blinka, eller att texten *"LARM"* skrivs ut på terminalen)
- Möjlighet att ställa tiden genom att läsa av två knappar inkopplade till DARMA-systemet. (ej för distansläge)
- Tidtagningsläge med start/stop och mellantid
- Utskrift av aktuell tid till terminalen
- Mjuk övergång mellan två siffror genom att lysdiodssegmentens intensitet ser ut att ändras långsamt. Detta kan göras genom att använda så kallad PWM-modulering (se Google). Det här är troligtvis den mest effektfulla och lärorika förändringen du kan göra, om än inte den lättaste. (ej för distansläge)

4. Sammanfattning

Vi det här laget har du tillverkat en fullt fungerande klocka som fungerar enbart med avbrott. Även om vi hade ett annat program igång skulle avbrottsingångarnas signaler se till så att klockan går rätt oberoende av vad processorn håller på med.

En fördel med avbrott är att vi kan låta yttre hårdvara skapa dessa med en mycket högre noggranhet än vad processorn själv skulle kunnat. Processorns möjlighet att mäta tid är begränsad av både processorklockans noggranhet och instruktionernas exekveringstid.

-o-O-o-

A. Schema på sjusegmentsmodulen