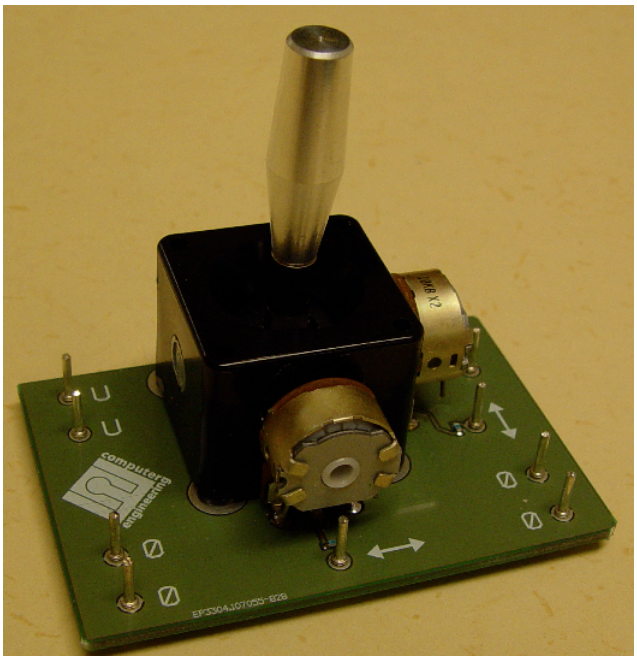
Datorteknik TSIU02 Lab 4
Spel — v0.3

Inledning



Denna laboration är ett projekt där du skall tillverka ett digitalt spel. Spelet går ut på att med en joystick leta upp målet som slocknar då din markör träffar målet. Som spelplan används en lysdiodmatris om 5x7 tecken. Målet markeras med en tänd lysdiod och din markör med en annan lysdiod.

Programstrukturen anges men det saknas flera viktiga rutiner. I programlistningen anges vad som skall hända i respektive rutin. Det är din uppgift att skriva färdigt programmet.

Obs! I laborationen finns inga förberedelseuppgifter angivna. Det betyder dock inte att förberedelser inte behövs. Tvärtom, i denna laboration är de mer nödvändiga än någonsin tidigare och allt måste förberedas. Det är upp till dig att komma till laborationen tillräckligt förberedd. Till laborationen **måste** du medföra konstruktionsunderlag, exempelvis strukturdiagram, programkod och kopplingsschema.

Du måste dessutom använda ett strukturerat angreppssätt i programmeringen och ett ingenjörsmässigt tänkande genom hela uppgiften. Det finns många fallgropar och förmågan att inse dessa i tid, och gå runt dem eller lösa dem, är en väsentlig del i den här laborationen.

Utrustning

Till den här laborationen behöver du använda ny hårdvara:

- Joystick
- Spelplan
- A/D-omvandlare
- Högtalare

Dessutom finns tryckknappar och lysdioder till ditt förfogande vid deltester.

Spelkrav

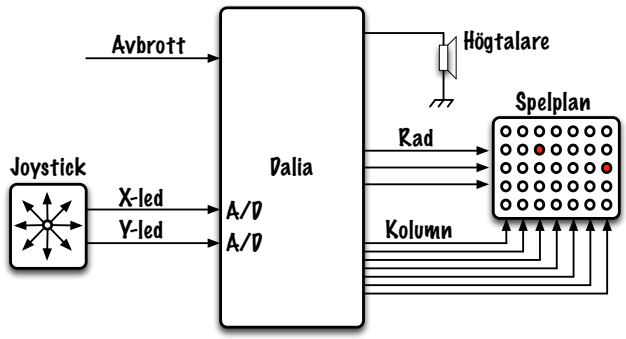
För godkänd laboration skall spelet uppfylla åtminstone följande spelkrav:

1. Vid spelstart skall ett mål placeras någonstans på spelplanens vänstra halva. Samtidigt skall en markör placeras i kolumnen längst till höger på spelplanen.
2. Spelarens uppgift är att flytta markören till målet.
3. När målet är täckt av markören skall träff signaleras med en ljudsignal i en högtalare.
4. Vid träff skall spelet startas om.
5. Subrutinen `RANDOM` skall anropas som angivet i koden, dvs med förberedda argument på stacken.

Hårdvara

Blockschema

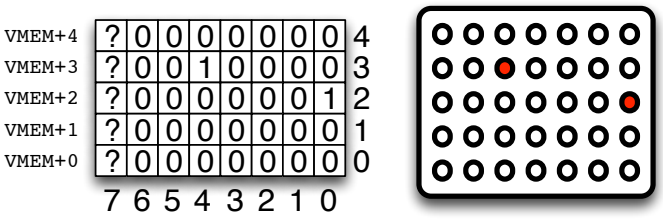
Spelet kräver joystick, A/D-omvandlare, spelplan och högtalare. Dessutom skall en yttre multiplexsignal initiera uppdatering av spelplanen. Figuren nedan anger hårdvarans principiella uppbyggnad.



Principschema över spelets hårdvara. joystickens läge omvandlas med inbyggd A/D-omvandlare och används av programmet för att styra den egna markören. "Målet" utgörs av en lysdiod i spelplanens vänstra halva. Vid träff skall ett ljud höras ur högtalaren.

Spelplan

Spelplanen återspeglar innehållet i ett videominne, `VMEM`, om 5 bytes. Planen är 5x7 lysdioder och använder en byte per rad i SRAM. Högsta biten i varje byte används inte. En tänd lysdiod markeras med en logisk etta och en släckt med logisk nolla:



Spelets koordinatsystem i x- och y-led är angivet i figuren. $x \in [0..6]$, $y \in [0..4]$.

Den egna positionen återfinns i två variabler `POSX` och `POSY`. Målets koordinater är på motsvarande sätt `TPOSX` och `TPOSY`

Spelplansuppdatering skall initieras av ett externt avbrott och utföras med en frekvens av 1 kHz. Avbrottsrutinen skall överföra videominnet till spelplanen, en rad per avbrott, med hjälp av multiplexning.

Joystick

Joysticken ger utspänningen 0–5 V i x- respektive y-led och skall avläsas med hjälp av processorns inbyggda A/D-omvandlare.

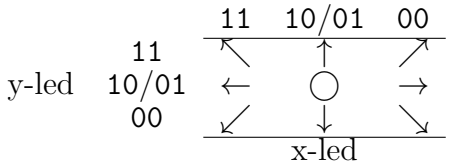
Enbart de två högsta bitarna, b_7b_6, av respektive omvandling används för den egna förflyttningen. Detta delar upp hela intervallet i fyra områden: `00`, `01`, `10`, `11`. Joysticken är i viloläge för värdena `01` och `10`.

De avlästa bitarna skall användas på följande sätt:

b_7b_6	riktning x-led	riktning y-led
11	vänster	upp
10	-	-
01	-	-
00	höger	ner

Med de två mest siginifikanta bitarna (b_7 och b_6) kan vi avgöra om spaken är i något av sina ändlägen eller i ett läge någonstans i mitten av sitt utslag.

Med denna konvention kan naturligtvis också kombinationer av dessa, exempelvis "snett nedåt höger", detekteras:



Med två A/D-omvandlarkanaler kan riktningarna upp, ner, höger, vänster såväl som diagonalriktningarna kännas av.

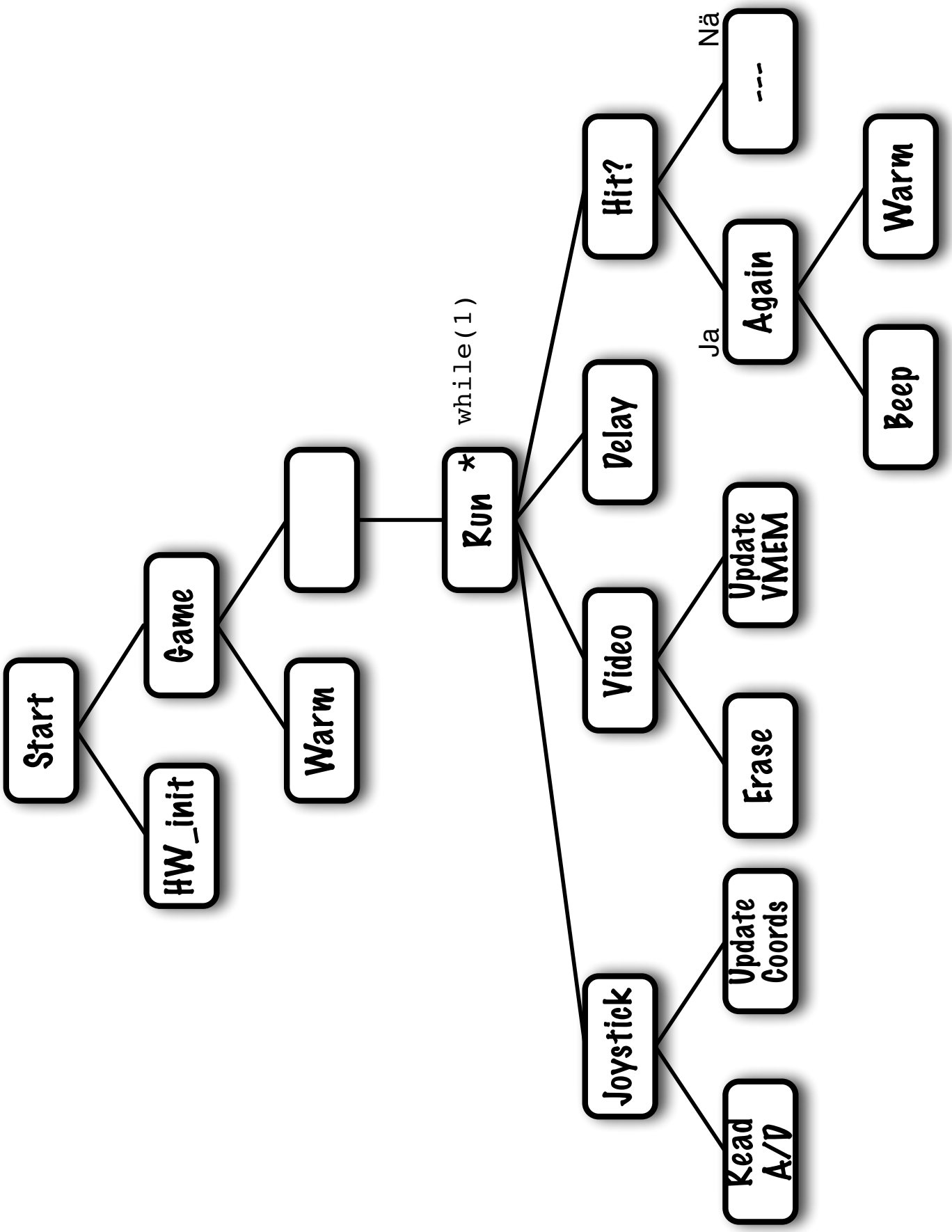
Strukturdiagram och programskal

På nästa sida ges en översikt över programmets programflöde. Programskalet följer i huvudsak det angivna flödet, speciellt används samma subrutinnamn.

Programskalet är avsiktligt inte komplett. "`***`" markerar områden som måste kompletteras. Du måste själv definiera de portar du vill använda. Rita ett schema! Du behöver skriva en del helt nya rutiner men använd programskalet som ledning. Du ska inte behöva avvika från flödet i programskalet.

Helt färdiga rutiner är `LIMITS` och `UPDATE` (med tillhörande subrutiner `POS_LIM`, `SETPOS` och `SETBIT`). Läs dem.

Översikt över programflödet (MUX-rutinen utelämnad)



Förberedelse- och laborationsuppgifter

Tänk på att förberedelserna, kod, strukturdiagram och elektriskt schema är direkta underlag för den senare programmeringen.

Var noggrann och ha schemat framför dig på laborationen.

Spelkonstruktionen skall uppdelas i två delar: Huvudprogrammet utgör spelmotorn och en avbrottsrutin sköter displayen.

Rita upp ett detaljerat **schema** över hur uppgiften kan lösas med befintlig hårdvara. Schemat skall vara så detaljerat som möjligt, dvs **varje** koppling mellan komponenterna skall redovisas. Konsultera bilagorna för mer information.

Studera och använd programlistningen. Vissa rutiner är färdigskrivna, andra är bara antydda och det är en del av förberedelserna att förstå och komplettera programkoden.

Laborationsuppgift Använd strukturdiagrammet och föreslagna assemblerrutiner för spelmotorn som underlag och komplettera tillämpliga delar. Du får själv skriva multiplexrutinen som läser av videominnet och uppdaterar spelplanen. Skriv in koden, kompilera, felsök.

Konstruktionsöverväganden

Som konstruktör av programmet är du ansvarig för allt. Målet med laborationen är att *i första hand* få igång spelet. Vägen dit behöver inte vara lika för alla.

Förslag till slumpgenerator Det är tillräckligt om vi kan ta fram två slumptal mellan 0 och 4.

Låt varje anrop till avbrottsrutinen för spelplansuppdatering räkna upp en byte, `RND`, i minnet. Det gör inget — i det här fallet är det faktiskt en fördel — om innehållet snabbt räknas upp som 254, 255, 0, 1, ... Använd de tre lägsta bitarna som slumptal. Om de tre lägsta bitarna ger ett tal som är större än vi önskar subtraherar vi 4 från det och använder resultatet som slumptal. Gör vi detta två gånger kan vi använda talen som koordinater i x- och y-led för ”målet”.

Du får naturligtvis hellre hitta på en egen metod att ta fram slumptalen!

Roligare spel Flera uppenbara tillägg till spelet kan göras. Skulle det till exempel vara svårt att införa följande förändringar?

- Få målet och/eller markören att blinka?
- ändra hastigheten hos markören?
- Låta målet flytta sig då markören är nära?
- Göra en roligare ljudeffekt med hjälp av PWM?
- Använd en inbyggd *timer* för att skapa avbrottssignalen.
- ...

Programskal

```
        ; --- lab4_skal.asm

        .equ    VMEM_SZ     = 5         ; #rows on display
        .equ    AD_CHAN_X   = 0         ; ADC0=PA0, PORTA bit 0 X-led
        .equ    AD_CHAN_Y   = 1         ; ADC1=PA1, PORTA bit 1 Y-led
        .equ    GAME_SPEED  = 70        ; inter-run delay (millisecs)
        .equ    PRESCALE    = 7         ; AD-prescaler value
        .equ    BEEP_PITCH  = 20        ; Victory beep pitch
        .equ    BEEP_LENGTH = 100       ; Victory beep length

        ; ---------------------------------------
        ; --- Memory layout in SRAM
        .dseg
        .org    SRAM_START
POSX:   .byte   1       ; Own position
POSY:   .byte   1
TPOSX:  .byte   1       ; Target position
TPOSY:  .byte   1
LINE:   .byte   1       ; Current line
VMEM:   .byte   VMEM_SZ ; Video MEMory
SEED:   .byte   1       ; Seed for Random

        ; ---------------------------------------
        ; --- Macros for inc/dec-rementing
        ; --- a byte in SRAM
        .macro INCSRAM  ; inc byte in SRAM
                lds     r16,@0
                inc     r16
                sts     @0,r16
        .endmacro

        .macro DECSRAM  ; dec byte in SRAM
                lds     r16,@0
                dec     r16
                sts     @0,r16
        .endmacro

        ; ---------------------------------------
        ; --- Code
        .cseg
        .org    $0
        jmp     START
        .org    INT0addr
        jmp     MUX


START:
        ***                     ; satt stackpekaren
        call    HW_INIT
        call    WARM
RUN:
        call    JOYSTICK
        call    ERASE
        call    UPDATE

***     Vanta en stund sa inte spelet gar for fort      ***

***     Avgor om traff                                  ***
```

```
        brne    NO_HIT
        ldi     r16,BEEP_LENGTH
        call    BEEP
        call    WARM
NO_HIT:
        jmp     RUN

        ; ---------------------------------------
        ; --- Multiplex display
        ; --- Uses: r16
MUX:

***     skriv rutin som handhar multiplexningen och     ***
***     utskriften till diodmatrisen. Oka SEED.         ***

        reti

        ; ---------------------------------------
        ; --- JOYSTICK Sense stick and update POSX, POSY
        ; --- Uses:
JOYSTICK:

***     skriv kod som okar eller minskar POSX beroende  ***
***     pa insignalen fran A/D-omvandlaren i X-led...    ***

***     ...och samma for Y-led                          ***

JOY_LIM:
        call    LIMITS          ; don't fall off world!
        ret

        ; ---------------------------------------
        ; --- LIMITS Limit POSX,POSY coordinates
        ; --- Uses: r16,r17
LIMITS:
        lds     r16,POSX        ; variable
        ldi     r17,7           ; upper limit+1
        call    POS_LIM         ; actual work
        sts     POSX,r16
        lds     r16,POSY        ; variable
        ldi     r17,5           ; upper limit+1
        call    POS_LIM         ; actual work
        sts     POSY,r16
        ret

POS_LIM:
        ori     r16,0           ; negative?
        brmi    POS_LESS        ; POSX neg => add 1
        cp      r16,r17         ; past edge
        brne    POS_OK
        subi    r16,2
POS_LESS:
        inc     r16
POS_OK:
        ret
```

```
        ; ---------------------------------------
        ; --- UPDATE VMEM
        ; --- with POSX/Y, TPOSX/Y
        ; --- Uses: r16, r17, Z
UPDATE:
        clr     ZH
        ldi     ZL,LOW(POSX)
        call    SETPOS
        clr     ZH
        ldi     ZL,LOW(TPOSX)
        call    SETPOS
        ret

        ; --- SETPOS Set bit pattern of r16 into *Z
        ; --- Uses: r16, r17, Z
        ; --- 1st call Z points to POSX at entry and POSY at exit
        ; --- 2nd call Z points to TPOSX at entry and TPOSY at exit
SETPOS:
        ld      r17,Z+          ; r17=POSX
        call    SETBIT          ; r16=bitpattern for VMEM+POSY
        ld      r17,Z           ; r17=POSY Z to POSY
        ldi     ZL,LOW(VMEM)
        add     ZL,r17          ; Z=VMEM+POSY, ZL=VMEM+0..4
        ld      r17,Z           ; current line in VMEM
        or      r17,r16         ; OR on place
        st      Z,r17           ; put back into VMEM
        ret

        ; --- SETBIT Set bit r17 on r16
        ; --- Uses: r16, r17
SETBIT:
        ldi     r16,$01         ; bit to shift
SETBIT_LOOP:
        dec     r17
        brmi    SETBIT_END      ; til done
        lsl     r16             ; shift
        jmp     SETBIT_LOOP
SETBIT_END:
        ret

        ; ---------------------------------------
        ; --- Hardware init
        ; --- Uses:
HW_INIT:

***     Konfigurera hardvara och MUX-avbrott enligt     ***
***     ditt elektriska schema. Konfigurera             ***
***     flanktriggat avbrott pa INT0 (PD2).             ***

        sei                     ; display on
        ret
```

```
        ; ---------------------------------------
        ; --- WARM start. Set up a new game.
        ; --- Uses:
WARM:

***     Satt startposition (POSX,POSY)=(0,2)            ***

        push    r0
        push    r0
        call    RANDOM  ; RANDOM returns TPOSX, TPOSY on stack

***     Satt startposition (TPOSX,TPOSY)                ***

        call    ERASE
        ret

        ; ---------------------------------------
        ; --- RANDOM generate TPOSX, TPOSY
        ; --- in variables passed on stack.
        ; --- Usage as:
        ; ---   push r0
        ; ---   push r0
        ; ---   call RANDOM
        ; ---   pop TPOSX
        ; ---   pop TPOSY
        ; --- Uses: r16
RANDOM:
        in      r16,SPH
        mov     ZH,r16
        in      r16,SPL
        mov     ZL,r16
        lds     r16,SEED

***     Anvand SEED for att berakna TPOSX               ***
***     Anvand SEED for att berakna TPOSY               ***

        ***             ; store TPOSX   2..6
        ***             ; store TPOSY   0..4
        ret

        ; ---------------------------------------
        ; --- ERASE videomemory
        ; --- Clears VMEM..VMEM+4
        ; --- Uses:
ERASE:

***     Radera videominnet                              ***

        ret

        ; ---------------------------------------
        ; --- BEEP(r16) r16 half cycles of BEEP-PITCH
        ; --- Uses:
BEEP:

*** skriv kod for ett ljud som ska markera traff        ***

        ret
```

-o-O-o-