Datorteknik TSEA82 + TSEA57 Fö9

A/D-omvandling

LINKÖPINGS UNIVERSITET

1

Datorteknik Fö9 : Agenda

- Repetition Preprocessor & macro
- A/D-omvandling
- Lab 4
- LAX
- Tid för frågor

2

Repetition Preprocessor & macro

LINKÖPINGS UNIVERSITET

Preprocessor och kompilering

Preprocessorn förstår ett antal direktiv, som alltså inte är egentlig assemblerkod utan just direktiv för att definiera, strukturera och tolka den övriga programkoden.

Preprocessordirektiven finns för att man enklare och tydligare ska kunna skriva och strukturera sin programkod.

Dom är egentligen inte nödvändiga för att åstadkomma det som programmet ska göra, men underlättar kodandet och gör det tydligare och lättare att läsa det resulterande programmet.

Direktiv	Namn	Betydelse
`.org`	*origin*	Skriv här (adress)
`.byte`	*byte*	Reservera byte i SRAM
`.dseg`	*data segment*	Följande gäller SRAM
`.cseg`	*code segment*	Följande gäller programminnet
`.eseg`	*extra segment*	Följande gäller EEPROM
`.def`	*define*	Döp register till namn
`.equ`	*equate*	Döp konstant
`.db`	*define byte*	Skriv följande *byte* (8-bit) i minnet
`.dw`	*define word*	Skriv följande *word* (16-bit) i minnet
`.macro`	*macro*	"copy-paste" av följande
`.endmacro`	*endmacro*	...avsluta ett macro
`<< n`	*shift left*	vänsterskift n bitar
`&, |, ^`	*logical AND, OR, XOR*	bitvis OCH, ELLER, XOR
`+, -, *, /`	*arithmetic*	som förväntat
`HIGH, LOW`	*high low*	ger höga resp låga delen av följande uttryck

LINKÖPINGS UNIVERSITET

Preprocessordirektiv

.org sätter kompilatorn (dvs kodgenereringen) till en specifik adress i SRAM- eller Flash-minnet.
Den adressen räknas automatiskt upp vid den fortsatta kodgenereringen.

.cseg anger att efterföljande kod ska hamna i programminnet.

.db definierar tabellvärden i programminnet (Flash)

```
    .cseg                   ; default
    .org $0000
    jmp  START
    ;
    ;    avbrottsvektorer
    ;
    .org INT_VECTORS_SIZE   ; 52
TAB: .db  1, 2, 3, 4
START:
    ;    programstart
```

Inte körbar kod, dvs exekveringen får inte komma hit.

Preprocessordirektiv

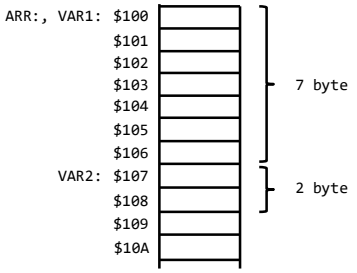
.org sätter kompilatorn (dvs kodgenereringen) till en specifik adress i SRAM- eller Flash-minnet.
Den adressen räknas automatiskt upp vid den fortsatta kodgenereringen.

.dseg anger att efterföljande definitioner ska använda SRAM (dataminnet).

.byte reserverar ett antal byte för variabler, bara det. Det går inte att tilldela värden till dessa variabler här.

```
    .dseg
    .org $100      ; adress $100 i SRAM
ARR:               ; ARR=$100, handtag till struct nedan
VAR1:    .byte 7   ; VAR1, adressen till 0-te byten av dessa 7
VAR2:    .byte 2   ; VAR2, adressem till första lediga efter VAR1

    .cseg
    ; till programminnet igen
    lds  r16,VAR1
    lds  r17,VAR2
    ...
```



Preprocessordirektiv

Följande visar på vilka möjligheter som preprocessorn ger oss när vi ska skriva kod.

Alla kodrader ger samtliga exakt samma resultat efter att preprocessorn gjort sitt.

```
ldi  r16,65
ldi  r16,$41
ldi  r16,0x41
ldi  r16,0b01000001
ldi  r16,(1<<6)|(1<<0)
ldi  r16,0xF1&0x4F
ldi  r16,'A'
ldi  r16,8*8+1
ldi  r16,HIGH($4122)
ldi  r16,LOW($2241)
ldi  r16,HIGH(16674)
ldi  r16,LOW(8769)
```

```
ldi  r16,0x41
```

Även följande ger samma resultat.

```
.equ N = $40
.def tmp = r16

ldi  tmp,N+1
ldi  tmp,N|(1<<0)
ldi  tmp,HIGH((N+1)<<8)
ldi  tmp,LOW(N|1)
```


Macron

Macron definierar ett kodstycke som ska kopieras in i koden. Det är inte detsamma som en subrutin, utan fungerar snarare som en ”stämpel” som preprocessorn använder när den genererar kod.

```
    .macro  PUSHZ
            push ZH
            push ZL
    .endmacro

    .macro  POPZ
            pop ZL
            pop ZH
    .endmacro
    ...
SUB:
    PUSHZ
    ...
    POPZ
    ret
```

```
SUB:
    push ZH
    push ZL
    ...
    pop ZL
    pop ZH
    ret
```

Macron

Macron kan även definieras med argument, vilket gör det lite mer användbart och dynamiskt.
Man skulle t ex kunna definiera den saknade instruktionen `ADDI`:

```
.macro  ADDI        ; macro med argumenten @0 och @1
        subi @0,-@1
.endmacro

...

ADDI    r20,3       ; r20 = r20 + 3
...
subi    r20,-3      ; r20 = r20 – (-3)
```

```
subi    r20,-3
...
subi    r20,-3
```


Macron

Macron ska dock användas sparsamt och ha väl valda namn. Annars blir koden snabbt obegriplig och svårläst, eftersom man egentligen skapar nya ord i grammatiken som inte tillhör språket från början. Dvs, den oinvigde måste själv göra översättningar av alla macron för att förstå koden.
Man får dessutom inte se vad macrot gör vid simulering, utan det bara utförs.

Utan att ha alla tidigare macro-definitioner i huvudet blir det svårt att veta vad följande kod gör:

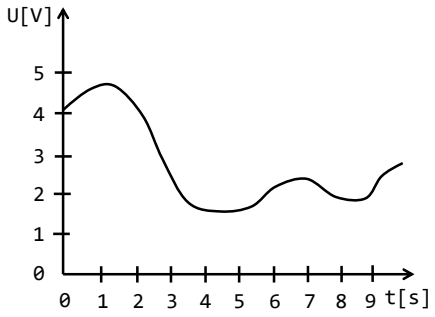
```
.macro  ...
.endmacro
...
LAST    r22
MIX     r17,r22
PUT     r17,4
BOX
LAST    Z+
...
```

Grundregel: Undvik macron om det inte är en *jättebra* idé och har en entydig och lättolkad funktion.
Använd macron sparsamt.

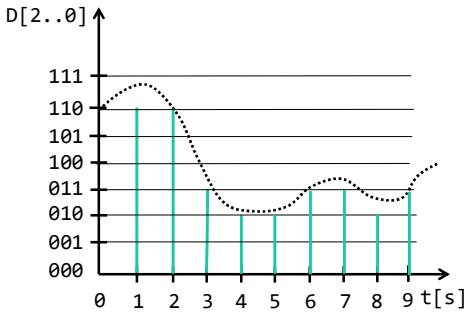
A/D-omvandling

A/D-omvandling

Analog.
Amplitudkontinuerlig och tidskontinuerlig



Digital.
Amplituddiskret och tidsdiskret



Kvantiseringseffekter med förlorad precision.

A/D-omvandling

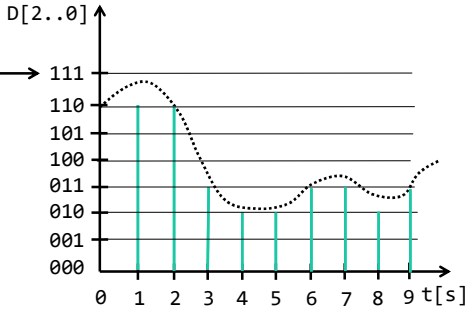
Vid A/D-omvandling jämförs den analoga insignalen med en referensspänning, AREF, som motsvarar taket, dvs den högsta tänkbara spänning som insignalen får ha.

Om insignalen uppnår AREF, så resulterar det i ett digitalt resultat med alla bitar 1-ställda.

En princip för A/D-omvandling är s k succesiv approximation. Då jämförs den analoga insignalen stegvis med respektive bits motsvarande spänning, med början på mest signifikant bit. Om jämförelsen är mindre, behålls biten (som 1:a), annars tas den bort, och sedan lägger man till spänningen för nästa bit.

För en 3-bitars omvandling skulle det då ta 3 cykler att komma fram till rätt digitalt värde.

Digital.
Amplituddiskret och tidsdiskret



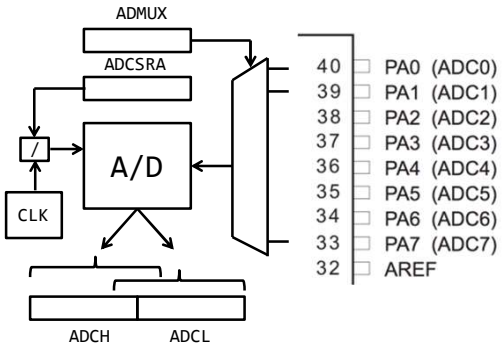
Kvantiseringseffekter med förlorad precision.

A/D-omvandling

ATMega328 har en 10-bitars A/D-omvandlare, vilket innebär att det digitala värdet har $2^{10} = 1024$ nivåer, dvs 0-1023 (0000000000_2-1111111111_2). Resultatet sparas i registerparet ADCH:ADCL, skiftat till vänster eller höger.

A/D-omvandlaren kan styras till att omvandla en av åtta analoga ingångar (ADC7-ADC0), via en multiplexer. ADMUX-registret väljer analog ingång, men även referensspänning, AREF, och resultatets skift.

ADCSRA är ett kontrollregister som aktiverar A/D-omv. Sätter omvandlingstakt m m.



A/D-omvandling, ADMUX

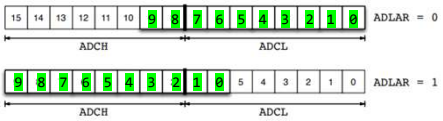
ADMUX väljer analog ingång, var referensspänningen kommer ifrån, samt justering av resultatet.

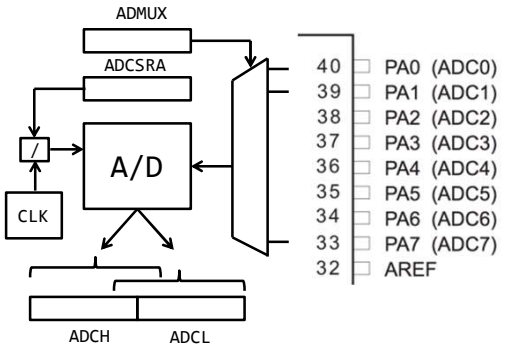
7	6	5	4	3	2	1	0	
REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX

Referensspänningen kan komma från extern ingång `AREF`, AV_{CC} eller från en intern spänning på `1.1`V.

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal V_{ref} turned off
0	1	AV_{CC} with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 1.1V Voltage Reference with external capacitor at AREF pin

Resultatet på `10` bitar ryms inte i ett register utan placeras till vänster eller höger inom registerparet ADCH:ADCL





A/D-omvandling, ADCSRA

ADCSRA aktiverar A/D-omvandlaren (`ADEN`), styr omvandlingstakten (`ADPSx`), startar en omvandling (`ADSC`) m m.

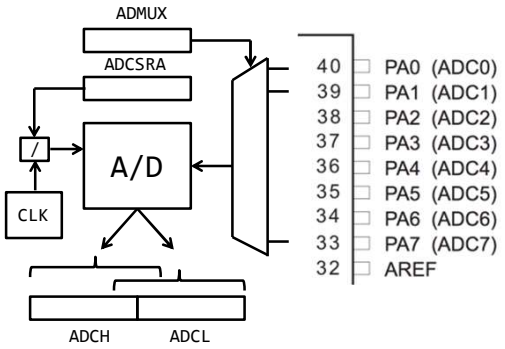
7	6	5	4	3	2	1	0	
ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA

Omvandlingstakten är max `15`kSPS (enl. datablad), och en omvandling tar `13` cykler. Det medför att processorklockan på `16` MHz måste skalas ned innan den används i A/D-omv.[1]

Table 21-5. ADC Prescaler Selections

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

En omvandling startas genom att `1`-ställa `ADSC`-biten, och `ADSC`-biten `0`-ställs automatisk när omvandlingen är klar.

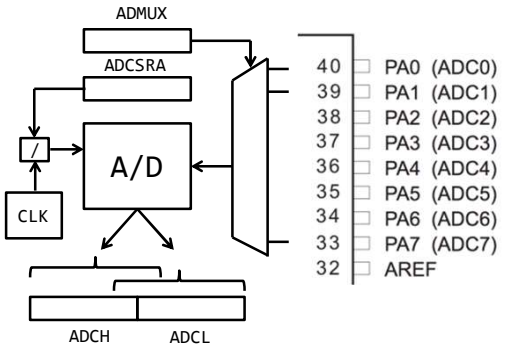


[1]Med `16` MHz, `13` cykler, `15000` kSPS måste `ADPSx` vara större än `82`: `16000000/13/15000 ≈ 82 => ADPSx=111`$_2$ `(dvs 128)`

A/D-omvandling, 10-bitars

Exempel: 10-bitars omvandling

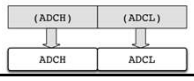
```
ADC10:
     ldi      r16,(1<<REFS0)     ; kanal 0, 5V ref, ADLAR=0
     sts      ADMUX,r16
     ldi      r16,(1<<ADEN) ; A/D enable, ADPSx=111
     ori      r16,(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)
     sts      ADCSRA,r16
ADC10_CONVERT:
     lds      r16,ADCSRA
     ori      r16,(1<<ADSC)
     sts      ADCSRA,r16         ; starta omvandling
ADC10_WAIT:
     lds      r16,ADCSRA
     sbrc     r16,ADSC           ; om 0-ställd, klar
     rjmp     ADC10_WAIT         ; annars vänta
     lds      r16,ADCL           ; obs, läs låg byte först
     lds      r17,ADCH           ; hög byte sedan
```



Resultatet återfinns i registerparet r17:r16

(Pågående omvandling)

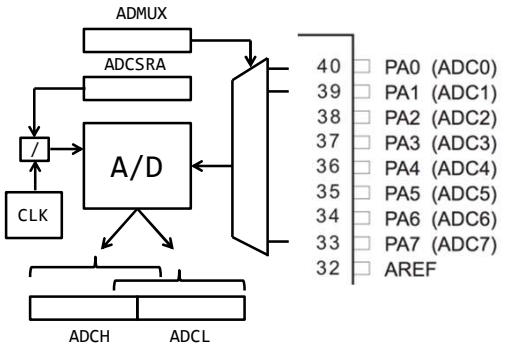
Föregående omvandling



A/D-omvandling, 8-bitars

Exempel: 8-bitars omvandling

```
ADC8:
     ldi      r16,(1<<REFS0)|(1<<ADLAR)   ; kanal 0, 5V ref
                                          ; left adjust
     sts      ADMUX,r16
     ldi      r16,(1<<ADEN) ; A/D enable, ADPSx=111
     ori      r16,(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)
     sts      ADCSRA,r16
ADC8_CONVERT:
     lds      r16,ADCSRA
     ori      r16,(1<<ADSC)
     sts      ADCSRA,r16         ; starta omvandling
ADC8_WAIT:
     lds      r16,ADCSRA
     sbrc     r16,ADSC           ; om 0-ställd, klar
     rjmp     ADC8_WAIT          ; annars vänta
     lds      r16,ADCH           ; en läsning av hög byte
```



Resultatet återfinns i registerparet r16.
Egentligen görs en 10-bitars omvandling, men endast de 8 höga bitarna används. Dvs, något sämre precision.

Labb 4

LINKÖPINGS UNIVERSITET

Lab4

Lab4 går ut på att skapa en enklare rad-editor.

På LCD:ns översta rad ska man med hjälp av knapparna `UP` och `DOWN` kunna bläddra fram nästa/föregående bokstav (`A-Z`), därefter med `LEFT` och `RIGHT` gå till föregående/nästa kolumn på raden.

Med knappen `SELECT` ska man kunna toggla bakgrundsbelysningen.

LCD:ns markör ska inte kunna gå utanför den översta raden.

På detta sätt ska man alltså kunna skriva in och även gå tillbaka med markören och redigera (ändra) redan inskriven text.

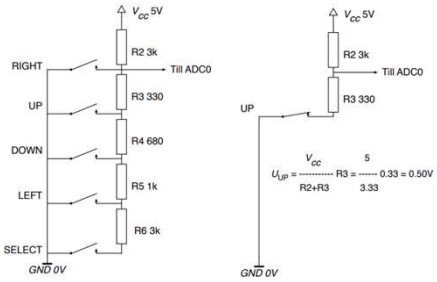
Knappen `RST` (reset) fungerar som omstart av processorn.

LINKÖPINGS UNIVERSITET

Lab4

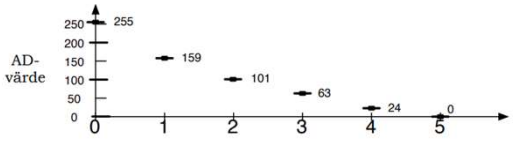
Knapparna är kopplade i en resistor-stege som kommer att ge en varierande analog utsignal, beroende på nedtryckt knapp.

T ex om `UP` trycks ned så kommer resistor-stegen att motsvara situationen i höger figur, och utsignalen blir ca `0.50` V.



Lab4

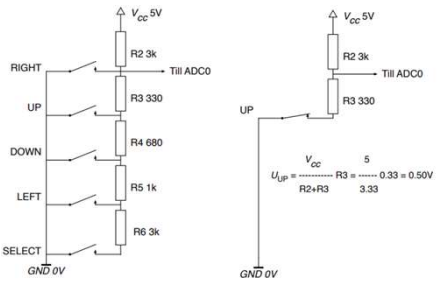
De 5 knapparna, samt ingen knapp nedtryckt, ger 6 olika analoga signaler motsvarande 6 olika digitala värden.



På grund av toleranser hos motstånden kan värdet variera, så räkna med att det kan ligga inom ett visst intervall.

Knapp	Ingen	SELECT	LEFT	DOWN	UP	RIGHT
Returvärde	0	1	2	3	4	5
Intervall	255–207	206–130	129–82	81–43	42–12	12–0

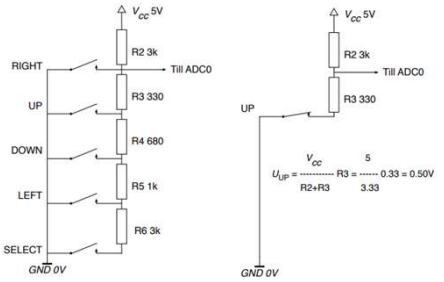
Börja med högsta intervallet, jämför med undre gränsen. Om insignalen är högre (eller lika) så är det den knappen, annars jämför med nästa undre gräns.



Lab4

Delmoment 1 : Skapa en utskriftsrutin, LCD_PRINT_HEX. Använd sedan den för att kontrollera vilka värden som A/D-omvandlaren ger, för olika knappar.

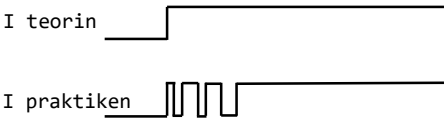
```
; Sub LCD_PRINT_HEX
; IN: r16, value
LCD_PRINT_HEX:
     call NIB2HEX
NIB2HEX:
     swap r16
     push r16
     andi r16,$0F
     ori  r16,$30       ; 00-09 => 30-39, 0A-0F => 3A-3F
     cpi  r16,$3A       ; 3A-3F?
     brlo NOT_AF        ; No, 30-39 (’0’-’9’)
     subi r16,-$07      ; Yes, 3A-3F => 41-46 (’A’-’F’)
NOT_AF:
     call LCD_ASCII
     pop  r16
     ret
```

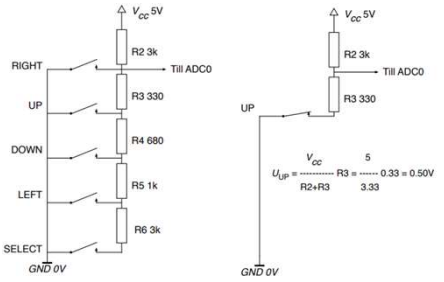


Lab4 : Kontaktstuds

En mekanisk strömbrytare har kontaktytor där det kan uppstå kontaktstuds vid tillslag och frånslag. För korrekt funktion kan det vara nödvändigt att vänta ut kontaktstuds innan avläsning.

```
KEY_READ:
     call     KEY                  ; read key
     tst      r16                  ; key=0?
     brne     KEY_READ             ; no, wait for key release
KEY_WAIT_FOR_PRESS:
     call     KEY                  ; read key
     tst      r16                  ; key=0?
     breq     KEY_WAIT_FOR_PRESS   ; yes, so wait for key
     call     delay                ; debounce...
     call     KEY                  ; read key again
     tst      r16                  ; key still pressed?
     breq     KEY_WAIT_FOR_PRESS   ; no, wait for key
     ; new key value available
     ret
```





LAX

LINKÖPINGS UNIVERSITET

25

LAX

-LAX:en föreslås gå 2/6 (alt 3/6 eller 4/6)
-Det kommer en gallup med förslag på tider
-LAX:en är 90 minuter
-Ingen anmälan
-LAX:en görs enskilt, dvs inget samarbete
-Man får använda alla tidigare programkod från labbar
-Frågor/”hjälp” via Teams, inlämning i Lisam
-Det finns övnings-LAX:ar på kurshemsidan
-Labbar + LAX = godkänd kurs

LINKÖPINGS UNIVERSITET

Tid för Frågor

LINKÖPINGS UNIVERSITET

27

Anders Nilsson

www.liu.se

LINKÖPINGS UNIVERSITET

28