Föreläsning 2 & 3

TSIU51

Niklaus Wirth, 1976:

Algorithms + Datastructures = Programs

```
C     A weird program for calculating Pi written in Fortran.
C     From: Fink, D.G., Computers and the Human Mind, Anchor Books, 1966.

      PROGRAM PI
      DIMENSION TERM(100)
      N=1
    3 TERM(N)=((-1)**(N+1))*(4./(2.*N-1.))
      N=N+1
      IF (N-101) 3,6,6
    6 N=1
    7 SUM98 = SUM98+TERM(N)
      WRITE(*,28) N, TERM(N)
      N=N+1
      IF (N-99) 7, 11, 11
   11 SUM99=SUM98+TERM(N)
      SUM100=SUM99+TERM(N+1)
      IF (SUM98-3.141592) 14,23,23
   14 IF (SUM99-3.141592) 23,23,15
   15 IF (SUM100-3.141592) 16,23,23
   16 AV89=(SUM98+SUM99)/2.
      AV90=(SUM99+SUM100)/2.
      COMANS=(AV89+AV90)/2.
      IF (COMANS-3.1415920) 21,19,19
   19 IF (COMANS-3.1415930) 20,21,21
   20 WRITE(*,26)
      GO TO 22
   21 WRITE(*,27) COMANS
   22 STOP
   23 WRITE(*,25)
      GO TO 22
   25 FORMAT('ERROR IN MAGNITUDE OF SUM')
   26 FORMAT('PROBLEM SOLVED')
   27 FORMAT('PROBLEM UNSOLVED', F14.6)
   28 FORMAT(I3, F14.6)
      END
```

"Spaghetti code"

Spaghetti code forts.

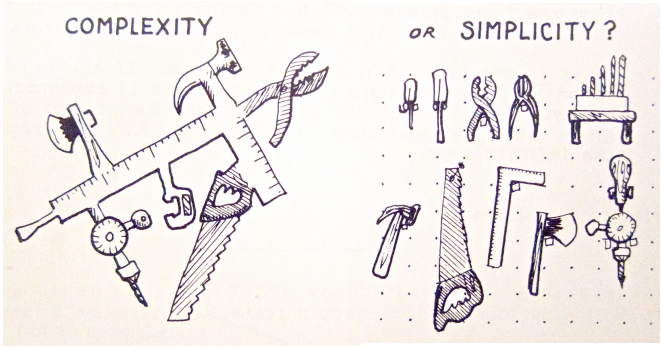
```
10 i = 0
20 i = i + 1
30 PRINT i; " squared = "; i * i
40 IF i >= 10 THEN GOTO 60
50 GOTO 20
60 PRINT "Program Completed."
70 END
```

Here is the same code written in a structured programming style:

```
10 FOR i = 1 TO 10
20     PRINT i; " squared = "; i * i
30 NEXT i
40 PRINT "Program Completed."
50 END
```

https://en.wikipedia.org/wiki/Spaghetti_code

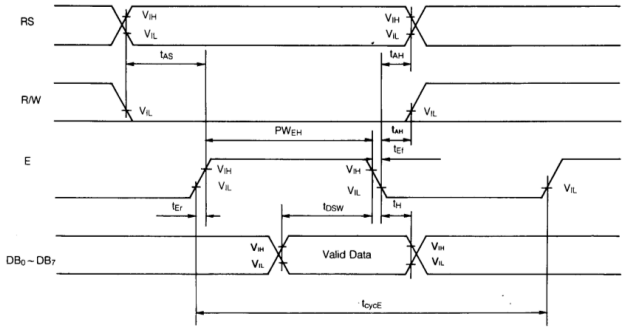
Vi har ett val!



Styrsekvens LCD

5.3 Timing Characteristics

Fig. 3 Write Operation Timing Diagram
(For data sent from the external microprocessor to the LCD unit)



Styrsekvens LCD

```
Hitta datat som ska skrivas
Sätt RS=0
Sätt RW=0
Lägg ut datat
Sätt E=1
Sätt E=0
Ta bort datat
Sätt RS=0
Sätt RW=1
```

Styrsekvens LCD

```
Data= 'G'          ; G      Sätt E=0
Sätt RS=0                   Ta bort datat
Sätt RW=0                   Sätt RS=0
Lägg ut datat               Sätt RW=1
Sätt E=1                    Data= 's'          ; s
Sätt E=0                    Sätt RS=0
Ta bort datat               Sätt RW=0
Sätt RS=0                   Lägg ut datat
Sätt RW=1                   Sätt E=1
Data= 'l'          ; l      Sätt E=0
Sätt RS=0                   Ta bort datat
Sätt RW=0                   Sätt RS=0
Lägg ut datat               Sätt RW=1
Sätt E=1                    Data= 's'          ; s
Sätt E=0                    Sätt RS=0
Ta bort datat               Sätt RW=0
Sätt RS=0                   Lägg ut datat
Sätt RW=1                   Sätt E=1
Data= 'a'          ; a      Sätt E=0
Sätt RS=0                   Ta bort datat
Sätt RW=0                   Sätt RS=0
Lägg ut datat               Sätt RW=1
Sätt E=1
```

Styrsekvens LCD

- Det blir lång kod. Total $5 \cdot 9 = 45$ programrader i exemplet ovan.
- Det är oöverskådligt. En längre textsträng, menyalternativ e dyl, blir flera sidor lång.
- Programmeringen blir lidande eftersom man tvingas tänka på flera nivåer samtidigt, så fort något skall hända måste man ner i varje detalj och gröta.
- Det drar programminne, som i varje fall när man programmerar mikrocontrollers, är en knapp resurs.
- Det blir svårändrad kod. Tänk om man vill införa en liten fördröjning mellan `Sätt E=1` och `Sätt E=0`. Ändringen måste införas på hela fem ställen. Inte bra.

Faktorisera

Faktorisera En kort blick på programmet ger att flera sekvenser återkommer. Genom att *faktorisera*[3] ut dessa till funktionsanrop eller subrutiner får vi mycket kortare kod. Sekvensen `Sätt E=1` och `Sätt E=0` är en tydlig kandidat för faktorisering. Vi skriver alltså en rutin, **`do_E`**, för att göra just detta:

```
do_E {
     Sätt E=1
     Sätt E=0
}
```

Faktorisera

```
do_LCD_write {
     Sätt RS=0
     Sätt RW = 0
     Lägg ut datat
     do_E
     Ta bort datat
     Sätt RS = 0
     Sätt RW = 1
}
```

Faktorisera

```
Data = 'G'  ; G
do_LCD_write
Data = 'l'  ; l
do_LCD_write
Data = 'a'  ; a
do_LCD_write
Data = 's'  ; s
do_LCD_write
Data = 's'  ; s
do_LCD_write
```

Parametrisera

```
    Text: 'Glass',0
LCD_ascii_print {
    Peka ut första tecknet
    Så länge tecken skiljt från 0 {
        Hämta tecknet
        do_LCD_write
        Peka ut nästa tecken
    }
}
```

It's easy to get so enamored of one's analytic tools that one forgets about the problem. The analyst must do more than carry out all possibilities of a problem to the nth degree, as I have seen authors of books on structured analysis recommend. That approach only increases the amount of available detail. The problem solver must also try to simplify the problem.

TIP 2.10

You don't understand a problem until you can simplify it.

If the goal of analysis is not only understanding, but simplification, then perhaps we've got more work to do.

Leo Brodie, Thinking Forth

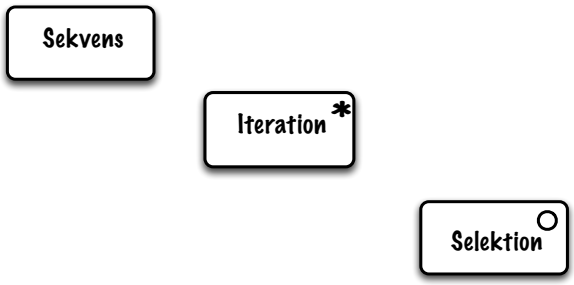
JSP, princip:

Data kan beskrivas i strukturdiagram.

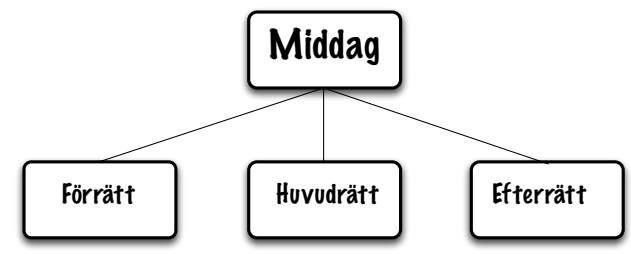
Programmet skrivs i samklang med datat.

Algorithms + Datastructures = Programs

Strukturdiagram Byggblock



Sekvens: först det ena, sen det andra och sen...

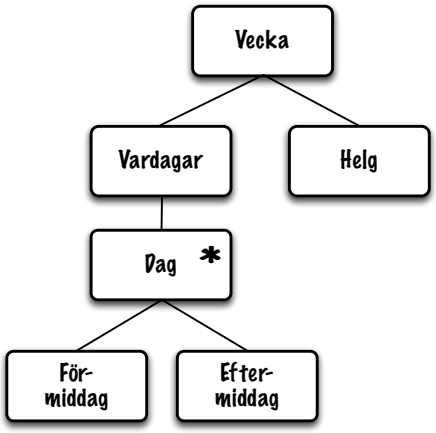


- Initiera hårdvara
- Definiera konstanter
- ...

- Utför databehandlingen
- Leverera delresultat
- ...

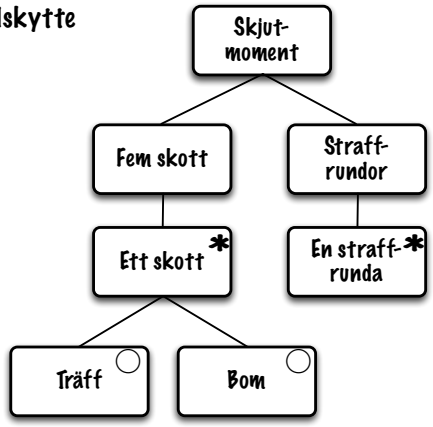
- Slutresultat
- Avsluta HW
- ...

Kombinerad sekvens och iteration



Selektion: det ena eller det andra eller ...

Skidskytte



JSP-Regler

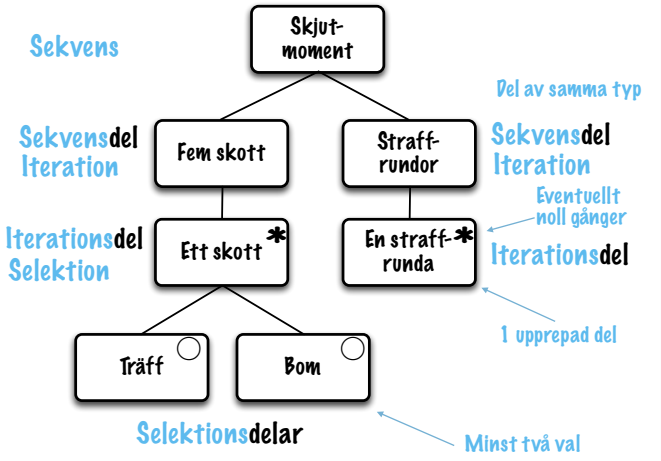
En komponent får endast ha delar av samma typ:
— En sekvens får inte bestå av iterationsdelar.
— En selektion får inte bestå av sekvensdelar osv.

En iteration får bara ha en (1) iterationsdel.

En iteration kan ske noll gånger:
— Tänk `while()` inte `if()`.

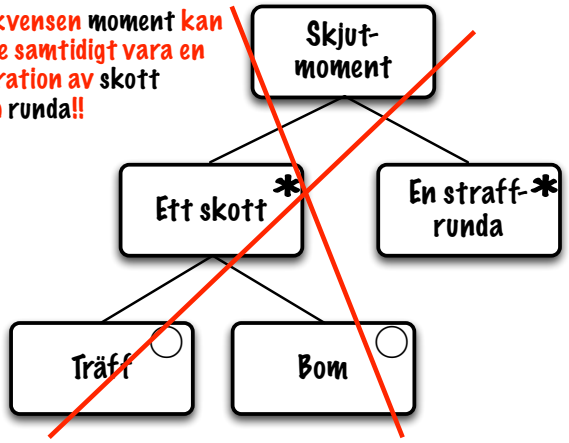
En selektion måste ha minst två delar:
— Ett val måste ha mer än ett alternativ.
— Ofta ett defaultalternativ också.

Korrekt enligt JSP

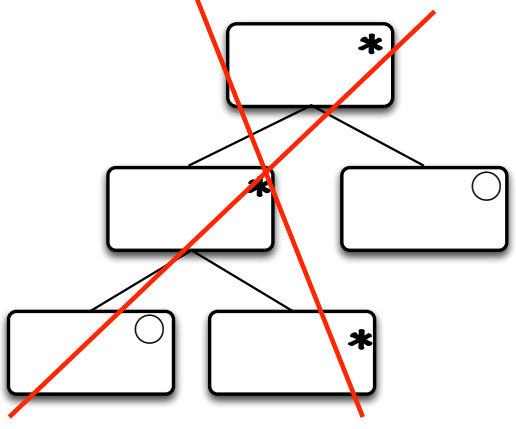


Inte korrekt enligt JSP

Sekvensen moment kan inte samtidigt vara en iteration av skott och runda!!



Inte JSP



Övergång till kod

Sekvens

```
    :
  call A
  call B
  call C
    :
```

Iteration

```
      :
AGAIN:
      cpi   r16,N
      brne  DONE
      call  B
      call  C
      jmp   AGAIN
DONE:
      :
```

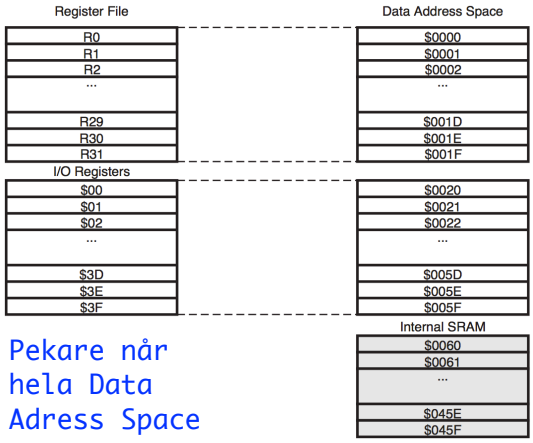
Selektion

```
      :
      cpi   r16,N
      brne  A1
      call  R
      jmp   DONE
A1:
      cpi   r16,M
      brne  A2
      call  S
      jmp   DONE
A2:
      :
DONE:
      :
```

Preprocessordirektiv

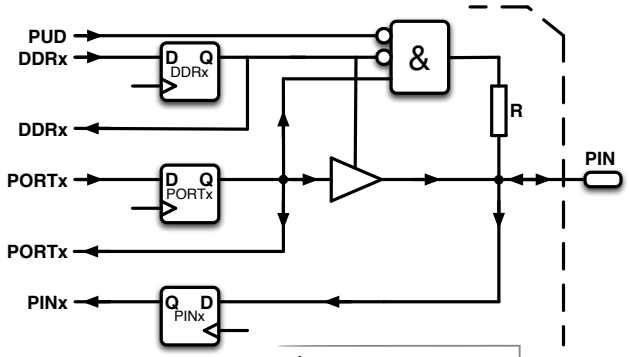
- .cseg Code Segment
- .dseg Data Segment
- .org Origin
- .db Define Byte
- .dw Define Word
- .byte Byte Array
- .def Define Symbol
- .equ Equate
- .macro Macro (!)

Minnesupplägg ATmega16



Pekare når hela Data Adress Space

I/O-port ATmega8/16




```
"in    r17,PINA
  :
 out   PORTA,r16"
```

Läsbar Kod

- Courier
- Parametrar
- .def/.equ
- Indentering
- Subrutiner
- Kommentarer

```
        ; ---------------------------------------
        ; --- UPDATE VMEM
        ; --- with POSX/Y, TPOSX/Y
        ; --- Uses r16, r17
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
        ; --- 1st call Z points to POSX at entry and POSY at ex
        ; --- 2nd call Z points to TPOSX at entry and TPOSY at
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          ; *(VMEM+T/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
```

Läsbar Kod

para-metrar

```
        ; --- lab4spel.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
```

man ska inte behöva peta här nere!

Läsbar Kod

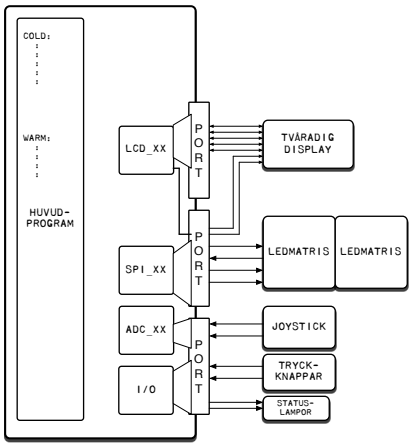
```
; ---------------------------------------
; --- Macros for inc/dec-rementing
; --- a byte in SRAM
.macro INCSRAM  ; inc byte in SRAM
        lds     r16,@0
        inc     r16
        sts     @0,r16
.endmacro

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

Minimera sidoeffekter

Lokala variabler på stacken

```
        ; --- Tidsavbrott
        ; --- Tiden i minnet MmSs där TIME: pekar  på "s"
INT_SECONDS:                    <---En entry per subrutin!
        push    r16
        in      r16,SREG
        push    r16
        push    XH
        push    XL
        clr     XH
        ldi     XL,TIME ; start with seconds
          :
          :                     Hopp enbart INOM subrutin,
INT_SEC_DONE:                           INTE mellan
        pop     XL
        pop     XH
        pop     r16
        out     SREG,r16
        pop     r16
        reti                    <---En exit per subrutin!
```

Minimera sidoeffekter



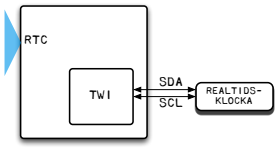
All hårdvara nås via drivrutin!

Parameter-överföring till drivrutiner via stacken

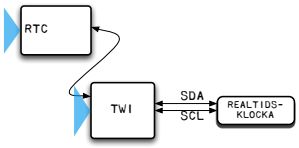
Minimera sidoeffekter

Strukturerad hårdvaruaccess

Stel lösning



Flexibel lösning



= Externt interface

Minimera sidoeffekter

Strukturerad hårdvaruaccess

```
open()  Initiera och aktivera
seek()  Adressera hårdvaran
read()  Hämta data
write() Skriv data
close() Avaktivera hårdvaran
```

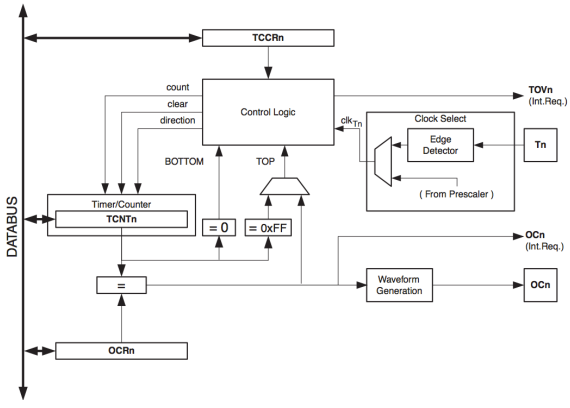
Minimera sidoeffekter

Strukturerad hårdvaruaccess

```
ADC_open()  ADEN, PS2..0,…
ADC_seek()  Välj ADC-kanal dvs sätt ADMUX
ADC_read()  Läs av ADCH
ADC_write() – (Inte tillämpbart)
ADC_close() A/D enable = 0 i ADCR

      :
    ldi     r16,AD_CHANNEL3
    push    r16
    call    ADC_read
    pop     r20
      :
```

Interna Timers Allmänt

Figure 27. 8-bit Timer/Counter Block Diagram

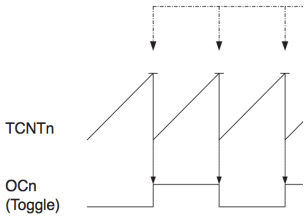


Intern Timer0 (8 bit)

- Timern räknar från BOTTOM till TOP
- Räknarvärdet är TCNT0
- Ger avbrott vid omslaget till BOTTOM
- Påverkar Output Capture-utgången OC0

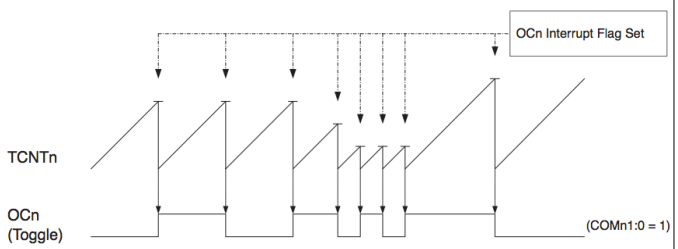
Mode NORMAL:

BOTTOM = $00 –> TOP = MAX = $FF



Mode CTC (Clear timer on compare):

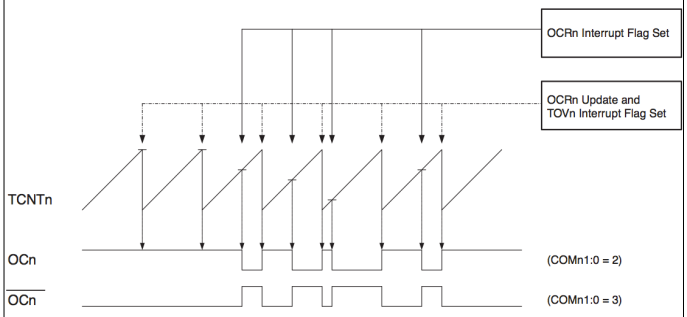
BOTTOM = $00 –> TOP = OCR0



Mode Fast PWM:

BOTTOM = $00 –> MAX = $FF

OCR0 jämför



Konfiguration Timer0

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Force Output Compare
Waveform Generation Mode 0, bit 0
Compare Output Match Mode0, bit 1
Compare Output Match Mode0, bit 0
Waveform Generation Mode 0, bit 1
CS02,01,00 Clock Select

Table 38. Waveform Generation Mode Bit Description[1]

Mode	WGM01 (CTC0)	WGM00 (PWM0)	Timer/Counter Mode of Operation	TOP	Update of OCR0	TOV0 Flag Set-on
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR0	Immediate	MAX
3	1	1	Fast PWM	0xFF	TOP	MAX

Table 42. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.

<– Finns fler…

Konfiguration Timer0

Normal mode, prescale clk/64

```
ldi     r16,(1<<CS01)|(1<<CS00)
out     TCCR0,r16
```

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Output Capture Interrupt Enable Timer0
Timer Overflow Interrupt Enable Timer0

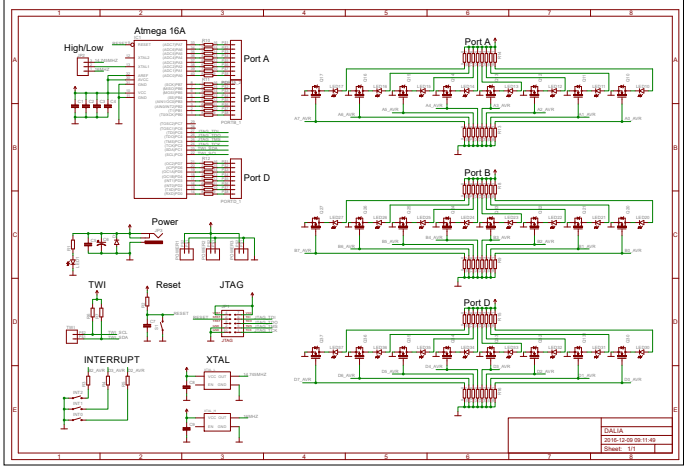
Overflow Interrupt Enable

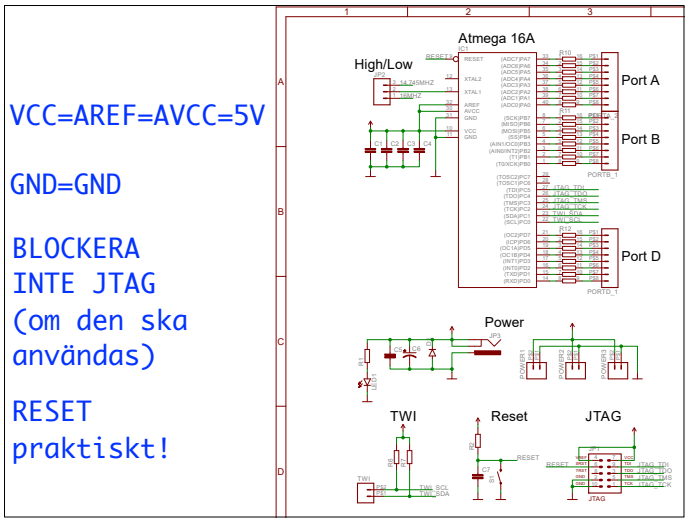
```
ldi  r16,(1<<TOIE0)
out  TIMSK,r16

sei          ; GO! (Avbrottsvektor  'OVF0addr')
```

Intern Timer 1 (16 bit)

- Huvudsakligen som timer0 fast 16 bitar
- Räknarvärdet är TCNT1H:TCNT1L
- Dubbelbuffrade 16-bitsregister:
  - Skriv HÖG byte före LÅG byte
  - Läs LÅG byte före HÖG byte
- Två utkanaler OCR1A, OCR1B
- Konfigurationsbitar utspridda…
- 16 olika moder!!

DALIA schema





VCC=AREF=AVCC=5V

GND=GND

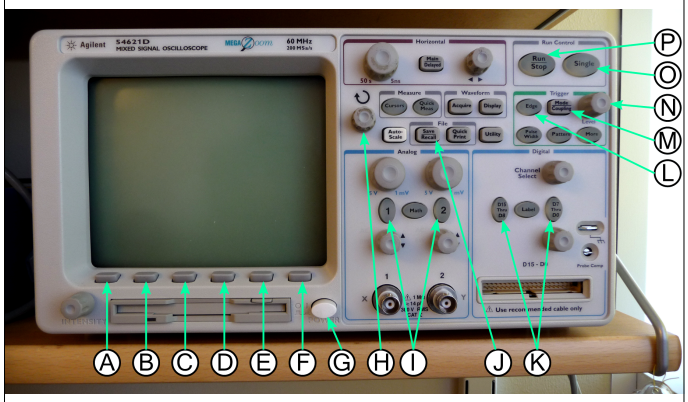
BLOCKERA
INTE JTAG
(om den ska
användas)

RESET
praktiskt!

LAB1 Digitaluret revisited

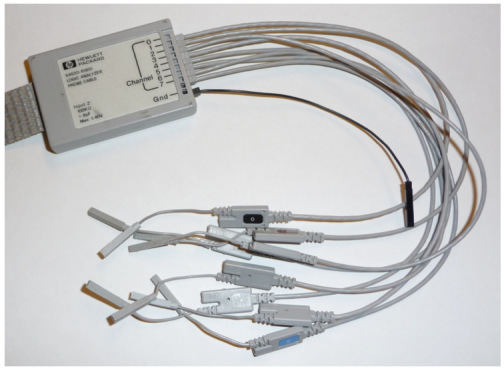
1. Snygga till koden
2. Använd timers för avbrotten

LAB2? Logikanalysator



LAB2? Digital prob

8 kanaler, 1 jord (GND)



Versionshantering med

git

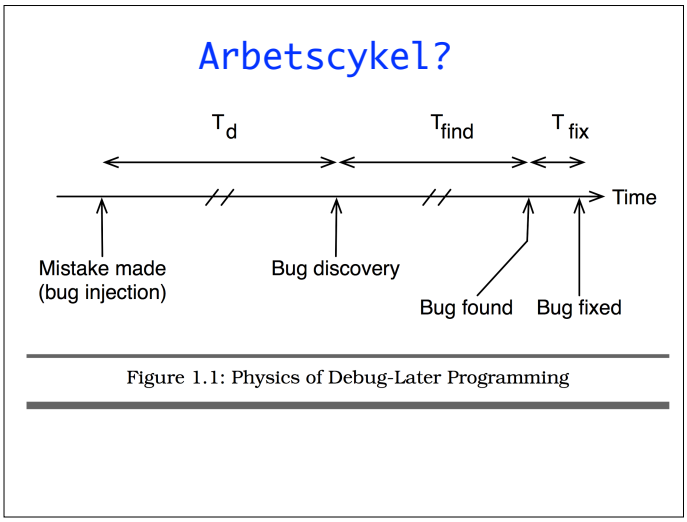


Figure 1.1: Physics of Debug-Later Programming

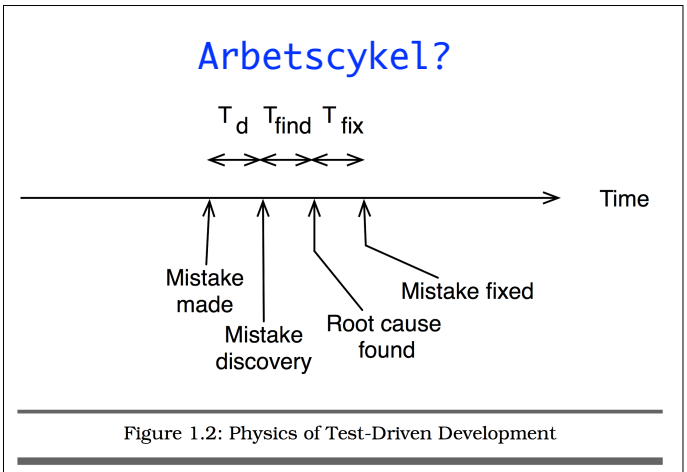


Figure 1.2: Physics of Test-Driven Development