

AVR Assembler

Özgun Mirtchev

March 22, 2016

Laborationsrapport för kursen Datorteknik 2016 VT

Universitetsadjunkt Kjell Mårdensjö

Örebro Universitet

Contents

1	Laboration 1: Lysdioder & Strömställare	4
1.1	Programmet Hello World	5
1.2	Assemblerprogram för att tända och släcka en lysdiod med en strömbrytare	5
1.3	Assemblerprogram för att tända och släcka en lysdiod med en switch	6
1.4	Ringräknare	7
1.5	Johnsonräknare	10
1.5.1	Kommentar	10
1.5.2	Kod	10
1.6	Lysdiodsmönster	12
2	Laboration 2: Subrutiner och aritmetik	14
2.1	Villkorliga programsatser i assembler	14
2.2	Elektronisk tärning	16
2.3	Förändringsräknare	17
2.4	Simulerad aritmetisk enhet (AU)	18
2.5	Några loopar	19
2.5.1	7-räknare	19
2.5.2	7- och 30-räknare	19
3	Laboration 3: Hårdvarunära programmering i C	21
3.1	C-program för att tända och släcka en lysdiod	21
3.2	Ringräknare i C	21
3.3	Spegling av en switches värde på en LCD-display	22
3.4	Elektronisk tärning med presentation på en LCD-display	24
3.5	Bitfäldsstruktur för att visa två tärningars värde på lysdioder	25
3.6	Mätning av exekveringstider med simulatören	25
3.6.1	Tid för att utföra multiplikation av två heltal med AVR-processor.	26
3.6.2	Utan och med volatile	26
3.6.3	Exekveringstider med en 8Mhz överklockad processor	27
3.6.4	Beräkningar på float med blandade datatyper	28
3.6.5	Resultat av tidsmätningar	29
3.7	Uppgift 7 - Strukturen sedd på maskinnivå som värdeparameter respektive retur-typ	29
3.8	Falsk elektronisk tärning	30
3.9	Kodlås	32
3.10	Avstudsning av en tryckknapp	35
3.10.1	Tillståndsdigram	35
3.10.2	C-kod	36
3.11	Trafikljusstyrning	37
3.11.1	Introduktion	37
3.11.2	Tillståndsdigram	38

3.11.3 C-Kod	39
4 Laboration 4: Objektorienterad programmering i C	42
4.1 Objektorienterad modellering av en mönstergenerator	42
4.2 Kast av tärningar	46
4.2.1 main-program	46
4.2.2 Dice-struct	47
4.2.3 Dice_LCD-struct	48
4.2.4 Dice_LEDS-struct	49
4.3 Avläsning av spänningen från en potentiometer	50
4.3.1 Main-program	50
4.3.2 Potentiometer-struct	50
4.3.3 LCD-struct	51
4.4 Styrning av ringräknarens blinkhastighet med hjälp av en potentiometer	53
4.5 Temperatursensor SMT160 med PWM-utgång	54
4.5.1 main-program	54
4.5.2 Sensor SMT160-struct	54
4.5.3 LCD-struct	56
4.6 Öka/Minska-knappar med acceleration	57
4.6.1 Main-program	57
4.6.2 Accelerator-struct	58
4.6.3 LCD-struct	60
4.7 Morse-kodlås	60
4.7.1 Main-program	60
4.7.2 Morse-struct	63
4.7.3 LCD-struct	65

1 Laboration 1: Lysdioder & Strömställare

I kommande koduppvisning kommer läsaren att märka främst två typer av funktioner, `init_func` och `loop_func`, som är generella namn på två funktioner som används i assembly-koden. Notera att dessa namn får och kan ändras till ens eget tycke men i denna laboration var funktionsnamnen oförändrade. C-koden ser ut som på nedanstående sätt och står för grunden av assembly-programmeringen. Istället för att skriva funktionerna i vanlig C ska dessa skrivas i assembly-kod som står med i slutet av respektive uppgift.

```
#include <avr/io.h>

int main(void)
{
    init_func();

    while (1)
    {
        loop_func();
        wait_milliseconds(300);
    }
}
```

I varje laboration kommer dessutom att fördefinierade värden för portarna och ingångarna att finnas:

```
;;;--- I/O-adresses for Port D ---
#define PIND    0x10
#define DDRD    0x11
#define PORTD   0x12

;;;--- I/O-adresses for Port C ---
#define PINC    0x13
#define DDRC    0x14
#define PORTC   0x15

;;;--- I/O-adresses for Port B ---
#define PINB    0x16
#define DDRB    0x17
#define PORTB   0x18

;;;--- I/O-adresses for Port A ---
#define PINA    0x19
#define DDRA    0x1A
#define PORTA   0x1B
```

1.1 Programmet Hello World

Denna uppgift var främst för att prova lite lätt assembly och därför finns ingen redovisning.

1.2 Assemblerprogram för att tända och släcka en lysdiod med en strömbrytare

Denna uppgift gick ut på att kunna tända en diod genom att trycka in respektive diod-knapp.

Algorithm 1 1.2 - Tända/släcka en lysdiod med strömbrytare

```
1      .data
2
3  lamps: .byte 0 ;; unsigned char lamps = 0;
4
5      .text
6      .global init_func
7
8  init_func:
9
10     ;; DDRA = 0x00
11     LDI    R20, 0x00
12     OUT    DDRA, R20
13
14     ;; DDRB = 0xFF;
15     LDI    R20, 0xFF
16     OUT    DDRB, R20
17
18     RET
19
20
21     .text
22     .global loop_func
23
24  loop_func:
25
26     ;; PORTB = PINA
27     IN     R20, PINA
28     OUT    PORTB, R20
29
30     RET
```

1.3 Assemblerprogram för att tända och släcka en lysdiod med en switch

Denna uppgift är har samma förutgrunder som föregående däremot är skillnaden att en switch används för att tända en lysdiod på ATmega32. När tredje switchen aktiveras lyser den första biten i registret som styr lysdioderna.

Algorithm 2 1.3 - Tända/släcka en lysdiod med switch

```
1 lamps: .byte 0      ;; unsigned char lamps = 0;
2
3         .text
4         .global init_func
5
6 init_func:
7         ;; DDRA = 0x00
8         LDI    R20, 0x00
9         OUT    DDRA, R20
10
11         ;; DDRB = 0xFF;
12         LDI    R20, 0xFF
13         OUT    DDRB, R20
14
15         ;; PORTB = 0xFF;
16         OUT    PORTB, R20
17
18         RET
19
20         .text
21         .global loop_func
22
23 loop_func:
24         ;; R20 = PINA & 0x08
25         IN     R20, PINA
26         COM    R20
27         ANDI   R20, 0b00001000
28
29         ;; R20 = R20 >> 3
30         LSR    R20
31         LSR    R20
32         LSR    R20
33
34         ;; R21 = PORTB & 0xFE
35         IN     R21, PORTB
36         COM    R21
37         ANDI   R21, 0xFE
38         OR     R20, R21      ; R20 = R20 | R21
39
40         COM    R20
41         OUT    PORTB, R20    ; PORTB = R20
42
43         RET
```

1.4 Ringräknare

Denna uppgift gick ut på att implementera en ringräknare. Processen för att skapa programmet följdes genom en figur från kurskompendiet som visar stegen för en lösning.

Som det syns i ovanstående figur finns det olika steg man kan följa. Notera att det finns alternativa lösningar till denna uppgift men denna valdes för att lättare kunna designa koden för ringräknaren. I denna lösning används processorns register medan de andra representerar lösningar som involverar en mönstertabell eller en if-else-sats. För enkelhetens skull gjordes denna uppgift med en lösning som använder processorns register genom T-bit utilisering.

Stegen som följdes demonstreras i nedanstående steg med tillhörande assembly-

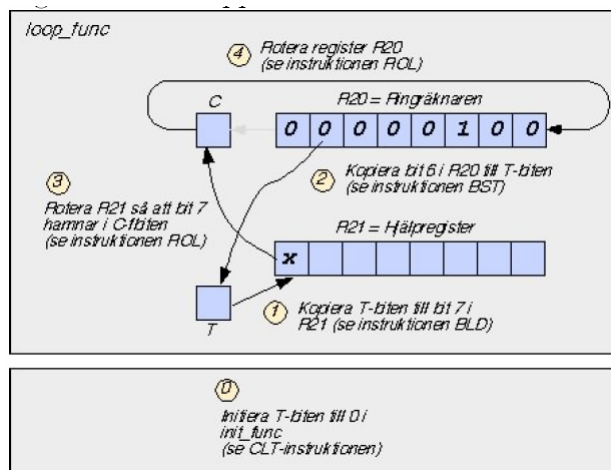


Figure 1: Ringräknare lösning genom register-användning

kod i algoritm-figuren.

1. I första steget initierades T-biten till 0 genom att använda CLT-instruktionen.
2. Andra steget inkluderade att kopiera T-biten till bit 7 i processorns R21-register som är användes som ett hjälpregister genom att använda instruktionen BLD.
3. Samma sak utfördes på registret R20, men i detta fall kopierades bit 6 till T-biten. R20 är ringräknaren, alltså registret som aktiverar lysdioderna.
4. Eftersom R21 fick bit 7 så roterades den så att den hamnar i C-biten som i nästa steg roterades in i ringräknaren-registret R20, båda använde instruktionen ROL.

Algorithm 3 1.4 - Ringräknare genom registeranvändning

```
1      .data
2
3      .text
4      .global init_func
5
6 init_func:
7
8      LDI R20, 0xFF
9      OUT DDRB, R20
10     CLT
11
12     RET
13
14     .text
15     .global loop_func
16
17 loop_func:
18
19     BLD R21, 7
20     BST R20, 6
21     ROL R21
22     ROL R20
23     OUT PORTB, R20
24
25     RET
```

1.5 Johnsonräknare

1.5.1 Kommentar

Koden som beskrivs nedan är i princip identisk mot ringräknar koden. Den enda skillnaden är att ett register inverteras i johnsonräknaren så att det fortsätter lysa även efter dioden har flyttat sig. I ringräknaren behövdes inte det eftersom det bara var en diod som skulle förflytta sig hela tiden.

1.5.2 Kod

Algorithm 4 1.5 - Johnsonräknare genom registeranvändning

```
1      .data
2
3      .text
4      .global init_func
5
6  init_func:
7
8      LDI R20, 0xFF
9      OUT DDRB, R20
10     CLT
11
12     RET
13
14     .text
15     .global loop_func
16
17  loop_func:
18
19     BLD     R21, 7
20     BST     R20, 6
21     COM     R21 ;; Detta steg gör ringräknaren till en ↔
22             johnsonräknare
23     ROL     R21
24     ROL     R20
25     OUT     PORTB, R20
26
27     RET
```

Algorithm 5 1.5 - Johnsonräknare genom if-else-användning

```
1 #define vjohn R20
2 #define one R21
3 #define vjohn_and R22
4 vjohn_temp: .byte 0
5
6     .text
7     .global init_func
8
9 init_func:
10     LDI vjohn, 0xFF
11     OUT DDRB, vjohn
12     LDI vjohn, vjohn_temp    ;; R20 = vjohn <=> R20 = 0
13     LDI one, 1    ;; R21 = 1
14     RET
15
16     .text
17     .global loop_func
18
19 loop_func:
20     LDS vjohn, vjohn_temp    ;; R20=vjohn
21     MOV vjohn_and, vjohn    ;; R22 = R20
22     ANDI vjohn_and, 0x80    ;; R22 & 0x80
23     BREQ if_equal
24     RJMP else_f
25
26 if_equal:
27     LSL vjohn    ;; R20 << 1
28     ADD vjohn, one    ;; R20 += R21
29     RJMP end_loop
30
31 else_f:
32     LSL vjohn    ;; R20 << 1
33     RJMP end_loop
34
35 end_loop:
36     STS vjohn_temp, vjohn
37     COM vjohn    ;; Inverterar för lysdioderna
38     OUT PORTB, vjohn
39     RET
```

1.6 Lysdiodsmönster

I denna uppgift skulle ett mönster skapas. Mönstret i fråga ser ut som en johnsonräknare som i cykler räknar inåt och sedan utåt. En representation kan fås genom nedanstående figur 2:

```
00000000
10000001
11000011
11100111
11111111
11100111
11000011
10000001
```

Figure 2: Lysdiodsmönster

För uppgiften användes en lista för att hålla reda på vilka lysdioder som ska lysa. Loop-funktionen gick igenom en for-loop liknande sats som aktiverade respektive position i listan.

Algorithm 6 1.6 - Lysdiodsmönster genom tabell

```
1      .data
2      .global LEDS
3
4  LEDS: .byte 0x00, 0x81, 0xC3, 0xE7, 0xFF, 0xE7, 0xC3, 0x81
5
6      .text
7      .global init_func
8
9  init_func:
10     ;; i = 0
11     LDI R28, 0x00
12     LDI R29, 0x00
13
14     ;; DDRB = 0xFF
15     LDI R24, 0xFF
16     OUT DDRB, R24
17
18     RET
19
20
21     .text
22     .global loop_func
23
24  loop_func:
25     ;; PORTB = LEDS[i]
26     LDI R30, lo8(LEDS)
27     LDI R31, hi8(LEDS)
28
29     ;; LEDS + i - R31:R30 + R29:R28
30     ADD R30, R28
31     ADC R31, R29
32
33     LD R24, Z      ;; Z = R31:R30
34     /*COM R24*/    ;; ignore
35     OUT PORTB, R24
36     ADIW R28, 0x01 ;; i += 1
37
38     ;; i = i & 0x07
39     ANDI R28, 0x07
40     ANDI R29, 0x00
41
42     RET
```

2 Laboration 2: Subrutiner och aritmetik

2.1 Villkorliga programsatser i assembler

Algorithm 7 2.1 - Villkorliga programsatser main-loop

```
.data
#define vjohn R20
#define vjohn_and R21
#define vring R22
vjohn_temp: .byte 0

.text
.global init_func
init_func:

    LDI R18, 0xFF
    OUT DDRB, R18
    LDI R17, 0x00
    OUT DDRA, R17
    LDI vjohn, vjohn_temp
    RET

.text
.global loop_func
loop_func:

    IN vjohn, PINA
    MOV R19, vjohn
    ANDI R19, 0x01
    BREQ ring_f
    RJMP john_f
```

Algorithm 8 2.1 - Villkorliga programsatser (Ring och johnräknare-funktioner)

```
ring_f:
    LSL vring          ;vring
    CPI vring, 0x00
    BREQ ring_eq
    RJMP ring_end

ring_eq:
    LDI vring, 0x01
    RJMP ring_end

ring_end:
    COM vring
    OUT PORTB, vring
    COM vring
    RET

john_f:
    LDS vjohn, vjohn_temp
    MOV vjohn_and, vjohn
    ANDI vjohn_and, 0x80
    BREQ john_if
    RJMP john_else

john_if:
    LSL vjohn
    INC vjohn
    RJMP john_end

john_else:
    LSL vjohn
    RJMP john_end

john_end:
    STS vjohn_temp, vjohn
    COM vjohn
    OUT PORTB, vjohn
    RET
```

2.2 Elektronisk tärning

Algorithm 9 2.2 - Elektronisk tärning

```
.data
pattern:
.byte 0x10, 0x82, 0x92, 0xC6, 0xD6, 0xEE

.text
.global init_func
init_func:

    LDI R20, 0xFF
    OUT DDRB, R20
    LDI R21, 0x00
    OUT DDRA, R21
    CLR R20
    RET

.text
.global loop_func
loop_func:

    IN R21, PINA
    ANDI R21, 0x01
    BREQ dice_update
    RJMP dice_end

dice_update:

    INC R20 ; counter++
    CPI R20, 0x06
    BREQ dice_if
    RJMP dice_end

dice_if:

    LDI R20, 0x00
    RJMP dice_end

dice_end:

    LDI R30, lo8(pattern)
    LDI R31, hi8(pattern)
    ADD R30, R20
    LD R24, Z
    COM R24
    OUT PORTB, R24
    RET
```

2.3 Förändringsräknare

Algorithm 10 2.3 - Förändringsräknare

```
.data
#define oldValue R22
#define newValue R23
counter: .byte 0

.text
.global init_func
init_func:

    LDI R20, 0xFF
    OUT DDRB, R20
    LDI R21, 0x00
    OUT DDRA, R21
    RET

.text
.global loop_func
loop_func:

    MOV oldValue, newValue
    /*IN R19, PINA
    ANDI R19, 0x01*/
    EOR R19, oldValue ;För automatisk räkning utan intryckning
    MOV newValue, R19
    CP oldValue, newValue
    BRNE loop_if
    RJMP loop_end

loop_if:

    LDS R18, counter
    INC R18
    STS counter, R18
    RJMP loop_end

loop_end:

    COM R18
    OUT PORTB, R18
    CLR R18
    RET
```

2.4 Simulerad aritmetisk enhet (AU)

Algorithm 11 2.4 - Simulerings-kod

```
.data

#define valueX R20
#define valueY R21

.text
.global init_func
init_func:

    LDI R23, 0xFF
    OUT DDRB, R23
    LDI R23, 0x00
    OUT DDRA, R23
    LDI R23, 0x00
    OUT DDRD, R23
    RET

.text
.global loop_func
loop_func:

    IN R22, PINA
    COM R22
    ANDI R22, 0xF0
    MOV valueX, R22
    IN R22, PINA
    COM R22
    LSL R22
    LSL R22
    LSL R22
    LSL R22
    MOV valueY, R22
    IN R23, PIND
    ANDI R23, 0x01
    CPI R23, 0x01
    BRNE Subtract_func
    RJMP Add_func

Subtract_func:

    SUB valueX, valueY
    IN R22, SREG
    COM valueX
    OUT PORTB, valueX
    RET

Add_func:

    ADD valueX, valueY
    ANDI valueX, 0xF0
    IN R22, SREG
    ANDI R22, 0x0F
    OR valueX, R22
    COM valueX
    OUT PORTB, valueX
    RET
```

2.5 Några loopar

2.5.1 7-räknare

Tanken med denna uppgift var att räkning skulle ske genom varje flank-detektering. Vid nedaktivering av en knapp för negativ flank och uppaktivering av en knapp för positiv flank. I nedastående kod räknar det däremot automatisk med hjälp av instruktionen EOR som har samma funktion som en XOR-operation.

Algorithm 12 2.5 - 7-räknare

```
.data
varX: .byte 0

.text
.global init_func

init_func:
    LDI R20, 0xFF
    OUT DDRB, R20
    RET

.text
.global loop_func

loop_func:
    CALL seven
    RET

seven:
    LDS R21, varX
    INC R21
    STS varX, R21

    CPI R21, 0x08
    BREQ reset

    LDI R24, 0
    LDI R25, 1
    CALL wait_milliseconds

    COM R21
    OUT PORTB, R21
    COM R21

    RJMP seven

reset:
    LDI R21, 0x00
    STS varX, R21
    CLR R21
    RET
```

2.5.2 7- och 30-räknare

Denna kod har lånat seven-funktionen som finns från förra deluppgiften och dessutom har en thirty-funktion lagts till som räknar ned från 30 efter att seven har körts tre gånger.

Algorithm 13 2.5 - 7 och 30-räknare

```
.data
varX: .byte 0
varY: .byte 0
.text
.global init_func
init_func:
    LDI R20, 0xFF
    OUT DDRB, R20
    RET
.text
.global loop_func
loop_func:
    CALL seven
    CALL seven
    CALL seven
    CALL thirty
    RET
seven:
    LDS R21, varX
    INC R21
    STS varX, R21
    ; varX = 7 ? reset : seven
    CPI R21, 0x08
    BREQ reset
    CALL display
    RJMP seven
thirty:
    LDS R21, varY
    DEC R21
    DEC R21
    STS varY, R21
    ; varY = 0 ? reset : thirty
    CPI R21, 0x00
    BREQ reset
    CALL display
    RJMP thirty
reset:
    LDI R21, 0x00
    STS varX, R21
    LDI R21, 0x1E
    STS varY, R21
    CLR R21
    RET
display:
    ; wait_milliseconds ~0.25s
    LDI R24, 0
    LDI R25, 1
    CALL wait_milliseconds
    COM R21
    OUT PORTB, R21
    COM R21
    RET
```

3 Laboration 3: Hårdvarunära programmering i C

I denna laboration kommer koden att främst skrivas i C. Många av laborationerna är nästan direkta översättningar från assembler-uppgifterna till C från förra laborationer.

3.1 C-program för att tända och släcka en lysdiod

Algorithm 14 3.1 - Tända/släcka en lysdiod C

```
int main(void)
{
    DDRA = 0x00;
    DDRB = 0xFF;

    while (1)
    {
        PORTB = PINA;
    }
}
```

3.2 Ringräknare i C

I uppgiften ska en ringräknare konstrueras som stannar vid knapptryckning (sw3) och fortsätter när man trycker igen.

Algorithm 15 3.2 - Ringräknare C

```
#include <avr/io.h>

void wait_milliseconds(int milliseconds);

int main(void)
{
    DDRA = 0x00;
    DDRB = 0xFF;

    char leds[8] = {0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80};
    unsigned char new_sw3, old_sw3, run;
    int i = 0;

    new_sw3 = old_sw3 = run = 1;

    while (1)
    {
        // switch
        old_sw3 = new_sw3;
        new_sw3 = (PINA & 0x08) == 0;
        if (old_sw3 && !new_sw3)
        {
            run = !run;
        }

        if (run)
        {
            PORTB = ~leds[i];
            i++;
            i = i & 0x07;
        }
        else
        {
            PORTB = 0xFF; //Släcka diod vid !run
        }
        wait_milliseconds(100);
    }
}
```

3.3 Spegling av en switchs värde på en LCD-display

Denna uppgift gick ut på att visa värden av den intryckta switchen på en LCD-display (exempel figur 3) . Koden blev uppdelade i funktioner för att göra det tydligare vad varje sektion av kod gör.



Figure 3: LCD-skärm för ATmega32

Algorithm 16 3.3 - Main-funktion LCD

```
lcd4 theDisplay;

void dec_to_bin(int sw_value, char * sw_bin);

int main(void)
{
    DDRA = 0x00;
    lcd4_init(&theDisplay, &PORTB, &DDRB, 4000, 100);

    int sw_value = 0;
    char s[20];

    while (1)
    {
        sw_value = PINA;
        print_to_lcd(s, sw_value);
    }
    return 0;
}
```

Algorithm 17 3.3 - Funktioner för LCD

```
void print_to_lcd(char s[], int sw_value)
{
    // PRINT HEX
    sprintf(s, "HEX = %02x", 0xFF - sw_value);
    lcd4_cup_row1(&theDisplay);
    lcd4_write_string(&theDisplay, s);

    // CALCULATE BINARY
    char sw_bin[9];
    dec_to_bin(sw_value, sw_bin);

    // PRINT BINARY
    sprintf(s, "BIN = %s", sw_bin);
    lcd4_cup_row2(&theDisplay);
    lcd4_write_string(&theDisplay, s);

    for (int i = 0; i <= 8; i++)
    {
        sw_bin[i] = 48;
    }
}

void dec_to_bin(int sw_value, char * sw_bin)
{
    int dec, i = 0;
    dec = 255 - sw_value;
    while(i < 8)
    {
        sw_bin[7-i] = dec % 2 + '0';
        dec /= 2;
        i++;
    }
    sw_bin[8] = '\0';
}
```

3.4 Elektronisk tärning med presentation på en LCD-display

Algorithm 18 3.4 - Elektronisk tärning på LCD

```
#include <avr/io.h>
#include <stdio.h>
#include <util/delay.h>
#include "lcd4.h"

lcd4 theDisplay;

const int DICE_MAX = 6;

int main(void)
{
    DDRA = 0x00;
    lcd4_init(&theDisplay, &PORTB, &DDRB, 4000, 100);

    int sw_button = 0;

    while (1)
    {
        sw_button = (PINA & 0x01) == 0;
        if (sw_button)
            print_to_lcd();
    }
    return 0;
}

void print_to_lcd()
{
    char s[20];
    int diceValue = rand() % DICE_MAX + 1;

    sprintf(s, "Dice Dec = %d", diceValue);
    lcd4_cup_row1(&theDisplay);
    lcd4_write_string(&theDisplay, s);
}
```

3.5 Bitfältsstrukturer för att visa två tärningars värde på lysdioder

Algorithm 19 3.5 - Bitfältsstrukturer - main.c

```
#include <avr/io.h>
#include <stdlib.h>
#include "Dice.h"

void wait_milliseconds(int);

int main(void)
{
    DDRA = 0x00;
    DDRB = 0xFF;

    bDicePORT.unusedOne = 1;
    bDicePORT.unusedTwo = 1;

    while (1)
    {
        int diceOneRandomValue = rand () % 6 + 1;
        int diceTwoRandomValue = rand () % 6 + 1;

        if (!(PINA & 0x08))
        {
            bDicePORT.diceOne = diceOneRandomValue;
            bDicePORT.diceTwo = diceTwoRandomValue;
        }
        wait_milliseconds(300);
    }
}
```

Algorithm 20 3.5 - Bitfältsstrukturer - Dice.h

```
#ifndef DICE_H_
#define DICE_H_

typedef struct
{
    unsigned char diceTwo : 3;    //Bit 0-2
    unsigned char unusedOne : 1;  //Bit 3 används ej
    unsigned char diceOne : 3;    //Bit 4-6
    unsigned char unusedTwo : 1;  //Bit 7 används ej.
} diceRegister;

#define bDicePORT (*(volatile diceRegister *) &PORTB)
#define bDiceDDR (*(volatile diceRegister *) &DDRB)
#define bDicePIN (*(volatile diceRegister *) &PINB)

#endif
```

3.6 Mätning av exekveringstider med simulatören

Följande kod analyserades för denna uppgift:

```

int main(void)
{
    gXi = 20;
    gYi = 30;
    gRi = gXi * gYi;
}

```

3.6.1 Tid för att utföra multiplikation av två heltal med AVR-processor.

Den beräknade tiden från simulationen framgav resultatet 22 μ s.

3.6.2 Utan och med volatile

Algorithm 21 3.6 - Assemblykod med volatile

```

{
    gXi = 20;
0000003E LDI R24,0x14      Load immediate
0000003F LDI R25,0x00      Load immediate
00000040 STS 0x0065,R25    Store direct to data space
00000042 STS 0x0064,R24    Store direct to data space
    gYi = 30;
00000044 LDI R24,0x1E      Load immediate
00000045 LDI R25,0x00      Load immediate
00000046 STS 0x0061,R25    Store direct to data space
00000048 STS 0x0060,R24    Store direct to data space
    gRi = gXi * gYi;
0000004A LDS R20,0x0064    Load direct from data space
0000004C LDS R21,0x0065    Load direct from data space
0000004E LDS R18,0x0060    Load direct from data space
00000050 LDS R19,0x0061    Load direct from data space
00000052 MUL R20,R18       Multiply unsigned
00000053 MOVW R24,R0        Copy register pair
00000054 MUL R20,R19         Multiply unsigned
00000055 ADD R25,R0         Add without carry
00000056 MUL R21,R18       Multiply unsigned
00000057 ADD R25,R0         Add without carry
00000058 CLR R1            Clear Register
00000059 STS 0x0063,R25      Store direct to data space
0000005B STS 0x0062,R24    Store direct to data space
}

```

Algorithm 22 3.6 - Assemblykod utan volatile

```
{
    gXi = 20;
0000003E LDI R24,0x14      Load immediate
0000003F LDI R25,0x00      Load immediate
00000040 STS 0x0065,R25    Store direct to data space
00000042 STS 0x0064,R24    Store direct to data space
    gYi = 30;
00000044 LDI R24,0x1E      Load immediate
00000045 LDI R25,0x00      Load immediate
00000046 STS 0x0061,R25    Store direct to data space
00000048 STS 0x0060,R24    Store direct to data space
    gRi = gXi * gYi;
0000004A LDI R24,0x58      Load immediate
0000004B LDI R25,0x02      Load immediate
0000004C STS 0x0063,R25    Store direct to data space
0000004E STS 0x0062,R24    Store direct to data space
}
```

Eftersom volatile tvingar programmet att köra utan kodoptimering sker beräkningen i assembly normalt **inte** till. Utan volatile optimeras den delen bort i assembly och resultatet läggs direkt in i registret.

3.6.3 Exekveringstider med en 8Mhz överklockad processor

Tidsmätningar utfördes på följande datatyper: int, long och float. Tiderna står till höger om varje beräkning i nedanstående kodram.

Algorithm 23 3.6 - Tidsmätningar på int, long och float

```
int main(void)
{
    IntTest();      // 39,25 us
    LongTest();     // 100,88 us
    FloatTest();    // 103,88 us
}

void IntTest()
{
    volatile int gXi = 0x0040, gYi = 0x0020, gRi = 0;
    gRi = gXi + gYi;    // 1,75 us
    gRi = gXi * gYi;    // 2,75 us
    gRi = gXi / gYi;    // 28,5 us
}

void LongTest()
{
    volatile long gXi = 0x0040L, gYi = 0x0020L, gRi = 0;
    gRi = gXi + gYi;    // 3,50 us
    gRi = gXi * gYi;    // 12,13 us
    gRi = gXi / gYi;    // 77,25 us
}

void FloatTest()
{
    volatile float gXi = 20.0, gYi = 30.0, gRi = 0;
    gRi = gXi + gYi;    // 15,38 us
    gRi = gXi * gYi;    // 18,50 us
    gRi = gXi / gYi;    // 62,00 us
}
```

3.6.4 Beräkningar på float med blandade datatyper

Algorithm 24 3.6 - Tidsmätningar på float

```
void FloatMixTest()
{
    volatile float gFi = 1.5;
    volatile float gRi = 0;

    // float = float * int;
    volatile int gXi = 0x0040;
    gRi = gFi * gXi;    // 26,25 us

    // float = float * long;
    volatile long gLi = 0x0020;
    gRi = gFi * gLi;    // 25,50 us

    // float = float * float;
    gRi = gFi * gFi;    // 20,88 us
}
```

3.6.5 Resultat av tidsmätningar

Denna simulering visar att datatypen int är snabbast att beräkna medan long och float kommer efter i beräkningstid. Addition är snabbare att beräkna, multiplikation är ungefär lika snabb som addition medan division är långsammast. Simuleringen i förra uppgiften visade också att det inte är så stora skillnader att utföra aritmetiska beräkningar mellan olika datatyper då det bara skiljde några mikrosekunder. Generellt sett verkar det som om float är den datatypen som tar längst tid att utföra beräkningar med. Detta kan bero på att flera instruktioner måste utföras i assembly för att göra samma beräkning, jämfört med till exempel int. Long har 64 bitar, medan float har 32-bitar, likadant som int, men skillnaden är att float har "decimalbitar" som kan försvåra beräkningarna någorlunda.

3.7 Uppgift 7 - Strukturen sedd på maskinnivån som värdeparameter respektive retur-typ

1. Struktens medlemmar läggs till i registren i minnesstacken i en sekventiell följd. Början av strukten lagras på en adress i minnet där också första medlemmen av strukten räknas, därefter kommer resterande medlemmar ordnade efter varandra i respektive address beroende på storleken av datamedlemmarna. Alltså om struktens början lagras på adress 0x40, lagras också den första medlemmen också på 0x40. Om andra medlemmen är en int så lagras denna på 0x42 då en int är två bytes.
2. En strukt returneras av en funktion genom att först skicka adressen av funktionens början och därefter struktens adress i jämförelse med funktionen.

3.8 Falsk elektronisk tärning

Algorithm 25 3.8 - Dice - main.c

```
#include <avr/io.h>
#include <stdlib.h>
#include "..\..\help_files\Dice.h"

void wait_milliseconds(int);
void ThrowDice(char, int *, char, int *);

enum { Normal, False } Dice;

int main(void)
{
    DDRA = 0x00;
    DDRB = 0xFF;

    int nFalseThrow = 0;
    int diceValue = 6; // Inverted value for diode display

    bDicePORT.unusedOne = 1;
    bDicePORT.unusedTwo = 1;
    bDicePORT.diceTwo = diceValue;
    bDicePORT.diceOne = diceValue;

    Dice = Normal;

    while (1)
    {
        char NormalThrow = !(PINA & 0x01);
        char FalseThrow = !(PINA & 0x10);

        ThrowDice(NormalThrow, &diceValue, FalseThrow, &nFalseThrow)
    }
}
```

Algorithm 26 3.8 - ThrowDice - function

```
void ThrowDice(char NormalThrow, int *diceValue,
char FalseThrow, int *nFalseThrow)
{
    switch(Dice)
    {
        case Normal:
        {
            if(NormalThrow)
            {
                if(*diceValue >= 1)
                {
                    bDicePORT.diceTwo = *diceValue;
                    bDicePORT.diceOne = *diceValue;
                    *diceValue--;
                    wait_milliseconds(200);
                }
                else
                {
                    *diceValue = 6;
                }
            }
            else if(FalseThrow)
            {
                Dice = False;
                *nFalseThrow = 0;
            }
        } break; // Normal
        case False:
        {
            if(NormalThrow)
            {
                *diceValue = 1;
                if(*nFalseThrow < 2)
                {
                    bDicePORT.diceTwo = *diceValue;
                    bDicePORT.diceOne = *diceValue;
                    *nFalseThrow++;
                    wait_milliseconds(200);
                }
            }
            else
            {
                *diceValue = 6;
                bDicePORT.diceTwo = *diceValue;
                bDicePORT.diceOne = *diceValue;
                Dice = Normal;
                *nFalseThrow = 0;
            }
        }
    } break; //False
    default:
    break;
    } // Switch
}
```

3.9 Kodlås

Algorithm 27 3.9 - Kodlås - main

```
#include <avr/io.h>

enum
{
    Closed,
    Open,
    PushKey,
    ReleaseKey
} Locker;

void TryUnlock(char, int *, int *, char, char, int *, const int);
void CheckIfWrongCombination(int*, char, int*, char, char, int*, const int);
int CheckTrueCombination(int, char, int*, char, char );

int main(void)
{
    DDRA = 0x00;
    DDRB = 0xFF;

    const int timeout = 20;
    int time_counter = 0;
    int iCode = 0;
    int code[6] = {1,2,3,1,1,2};
    Locker = Closed;

    while (1)
    {
        char eventKey1 = !(PINA & 0x80);
        char eventKey2 = !(PINA & 0x40);
        char eventKey3 = !(PINA & 0x20);
        PORTB = PINA;

        TryUnlock(eventKey1, code, &iCode, eventKey2, eventKey3, &time_counter, timeout);

        wait_milliseconds(50);
    }
}
```

Algorithm 28 3.9 - TryUnlock-function

```
void TryUnlock(char eventKey1, int * code, int *iCode,
char eventKey2, char eventKey3, int *time_counter, const int timeout)
{
    switch(Locker)
    {
        case Closed:
        {
            if((eventKey1 && (code[*iCode] == 1))
            || (eventKey2 && (code[*iCode] == 2))
            || (eventKey3 && (code[*iCode] == 3)))
            {
                *iCode++;
                Locker = PushKey;
            }
            PORTB = 254;
        } break; // Closed
        case Open:
        {
            if(*time_counter > timeout)
            {
                Locker = Closed;
                *time_counter = 0;
                *iCode = 0;
            }

            *time_counter++;
            PORTB = 255 - *time_counter; //20 sec
            wait_milliseconds(500);
        } break; // Open
        case PushKey:
        {
            if(!eventKey1
            && !eventKey2
            && !eventKey3)
            {
                Locker = ReleaseKey;
                *time_counter++;
            }
            if(*iCode >= 6)
            {
                Locker = Open;
                *time_counter = 0;
            }
        } break; // PushKey
        case ReleaseKey:
        {
            CheckIfWrongCombination(
                &*iCode, eventKey1, code,
                eventKey2, eventKey3,
                &*time_counter, timeout
            );

            *iCode = CheckTrueCombination(
                *iCode, eventKey1, code,
                eventKey2, eventKey3
            );

            PORTB = 254;
        } break; // ReleaseKey
        default:
        break;
    } // Switch
}
```

Algorithm 29 3.9 - CodeCheck-functions

```
void CheckIfWrongCombination(  
    int *iCode, char eventKey1, int * code,  
    char eventKey2, char eventKey3,  
    int *time_counter, const int timeout)  
{  
    if (*iCode < 3)  
    {  
        if ((eventKey1 && (code[*iCode] != 1))  
            || (eventKey2 && (code[*iCode] != 2))  
            || (eventKey3 && (code[*iCode] != 3))  
            || *time_counter > timeout)  
        {  
            Locker = Closed;  
            *time_counter = 0;  
            *iCode = 0;  
        }  
    }  
    else  
    {  
        if ((eventKey1 && (code[*iCode] != 1))  
            || (eventKey2 && (code[*iCode] != 2)))  
        {  
            Locker = Closed;  
            *time_counter = 0;  
            *iCode = 0;  
        }  
    }  
}  
  
int CheckTrueCombination(  
    int iCode, char eventKey1, int * code,  
    char eventKey2, char eventKey3)  
{  
    if (iCode < 3)  
    {  
        if ((eventKey1 && (code[iCode] == 1))  
            || (eventKey2 && (code[iCode] == 2))  
            || (eventKey3 && (code[iCode] == 3)))  
        {  
            iCode++;  
            Locker = PushKey;  
        }  
    }  
    else  
    {  
        if ((eventKey1 && (code[iCode] == 1))  
            || (eventKey2 && (code[iCode] == 2)))  
        {  
            iCode++;  
            Locker = PushKey;  
        }  
    }  
    return iCode;  
}
```

3.10 Avstudsning av en tryckknapp

3.10.1 Tillståndsdigram

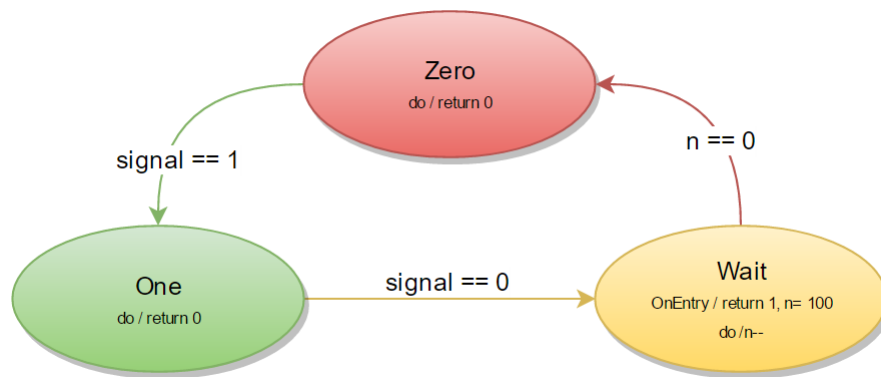


Figure 4: Tillståndsdigram för Avstudsning av en tryckknapp

3.10.2 C-kod

Algorithm 30 3.10 - Avstudsning - main

```
#include <avr/io.h>

static int n = 0;
static enum {Zero, Wait, One} Signal;

int main(void)
{
    DDRB = 0xFF;
    DDRA = 0x00;
    Signal = Zero;

    while (1)
    {
        char input = !(PINA & 0x01);
        PORTB = 255 - bounce(input);
        wait_milliseconds(100);
    }
}

int bounce(int signal)
{
    switch (Signal)
    {
        case Zero:
        {
            if (signal == 1)
            {
                Signal = One;
            }
            return 0;
        } break;
        case One:
        {
            if (signal == 0)
            {
                Signal = Wait;
                n = 10;
                return 1;
            }
            return 0;
        } break;
        case Wait:
        {
            n--;
            if (n == 0)
            {
                Signal = Zero;
                return 0;
            }
        } break;
        default:
            break;
    }
}
```

3.11 Trafikljusstyrning

3.11.1 Introduktion

I denna uppgift skulle trafikljus styras genom olika tillstånd. Lösningen genomfördes i C-kod men med objektorienterad stil, genom användning av structs. Följande figurer visar hur kretsens kopplingar såg ut och vilka lampor som representerade vilken riktning.

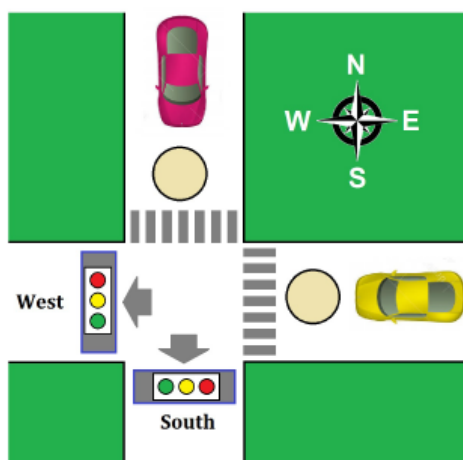


Figure 5: Trafikljus-korsning

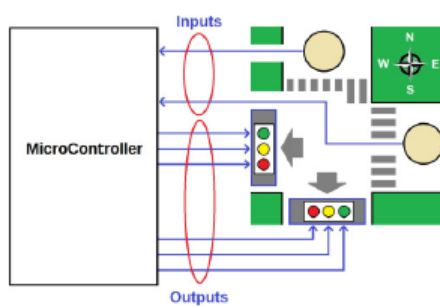


Figure 6: Systemdiagram

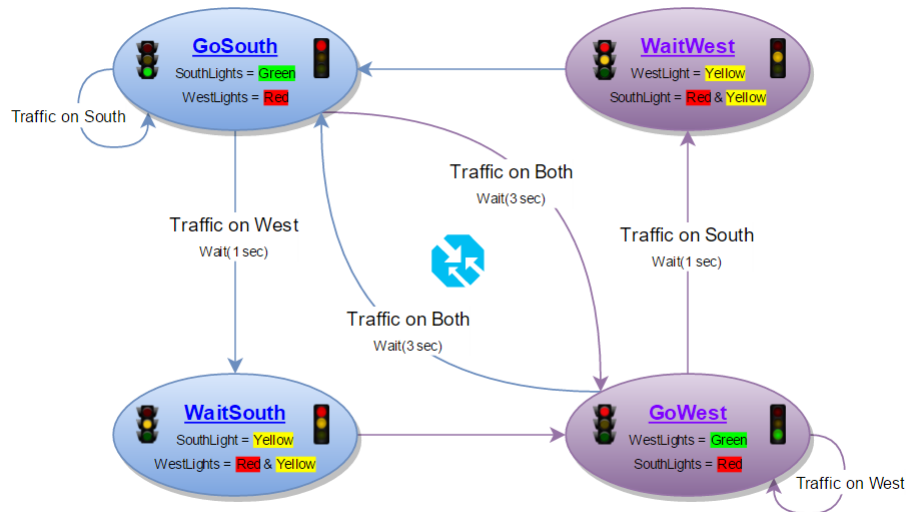


Figure 7: Tillståndsdigram Trafikljus

3.11.2 Tillståndsdigram

I figur 7 ser man den uppritade tillståndsdigrammet för denna uppgift. Trafik detektering sker via sensorer som man trycker på mikroprocessorn. Starttillstånd kommer att bli GoSouth, vilket innebär att det är trafik i södergående riktning, vilket då lyser upp den gröna lampan och låter den västgående riktningen vara röd. Mellanväxlingen mellan riktningarna sker på det viset att när trafik detekteras så påbörjas en nedräkning. Beroende på om det är en riktning som är trafikerad eller om båda riktningarna är trafikerade. När en riktning är trafikerad dras timern igång 2 sekunder snabbare än om båda riktningar skulle vara trafikerade. Detta för att vid lite trafik, inte låta trafikanterna vänta på ljusomslag. och vid mycket trafik, låta alla riktningarna få lika mycket tid att passera. Enligt tillståndsdigrammet visar de mittersta pilarna att omslaget sker direkt från grön till röd och vice versa, men detta är bara en ritning som indikerar en annan timer än de andra pilarna och för att förtydliga hur det går till. Transitionerna passerar alltid väntetillstånden. I nästkommande sidor visas koden för denna uppgift.

3.11.3 C-Kod

Algorithm 31 3.11 - TrafficLights - main.c

```
#include <avr/io.h>
#include "TrafficLights.h"

int main(void)
{
    DDRA = 0x00;
    DDRB = 0xFF;
    Traffic = GoSouth;

    while (1)
    {
        char TrafficOnSouth = !(PINA & 0x01);
        char TrafficOnWest = !(PINA & 0x02);
        switch (Traffic)
        {
            case GoSouth:
            {
                ActivateSouthLane(TrafficOnWest, TrafficOnSouth);

            } break;
            case WaitSouth:
            {
                TransitionSouthToWest();

            } break;
            case GoWest:
            {
                ActivateWestLane(TrafficOnSouth, TrafficOnWest);

            } break;
            case WaitWest:
            {
                TransitionWestToSouth();

            } break;
            default:
                break;
        }
    }
}
```

Algorithm 32 3.11 - Trafficlight.h Del I

```
#ifndef TRAFFICLIGHTS_H_
#define TRAFFICLIGHTS_H_

enum {
    GoSouth,
    WaitSouth,
    GoWest,
    WaitWest
} Traffic;

struct TrafficLights {
    char Red;
    char Yellow;
    char Green;
};

struct TrafficLights SouthTrafficLight = {0xFF - 0x20, 0xFF - 0x40, ↵
    0xFF - 0x80};
struct TrafficLights WestTrafficLight = {0xFF - 0x10, 0xFF - 0x08, ↵
    0xFF - 0x04};

// How to make this work?
// #define SouthLight (*(SouthTrafficLight)* &PORTB)
// #define WestLight  *(WestTrafficLight)* &PORTB)

int oneSec = 1000, threeSec = 3000; //Timer tid
```

Algorithm 33 3.11 - Trafficlight.h - Del II (Metoder)

```
void ActivateSouthLane(char TrafficOnWest, char TrafficOnSouth)
{
    PORTB = SouthTrafficLight.Green;
    PORTB += WestTrafficLight.Red;
    if(TrafficOnWest && !TrafficOnSouth)
    {
        wait_milliseconds(oneSec);
        Traffic = WaitSouth;
    }
    if(TrafficOnWest && TrafficOnSouth)
    {
        wait_milliseconds(threeSec);
        Traffic = WaitSouth;
    }
}

void TransitionSouthToWest()
{
    PORTB = WestTrafficLight.Red;
    PORTB += WestTrafficLight.Yellow;
    PORTB += SouthTrafficLight.Yellow + 1;
    wait_milliseconds(oneSec);
    Traffic = GoWest;
}

void ActivateWestLane(char TrafficOnSouth, char TrafficOnWest)
{
    PORTB = WestTrafficLight.Green;
    PORTB += SouthTrafficLight.Red - 1;
    if(TrafficOnSouth && !TrafficOnWest)
    {
        wait_milliseconds(oneSec);
        Traffic = WaitWest;
    }
    if(TrafficOnSouth && TrafficOnWest)
    {
        wait_milliseconds(threeSec);
        Traffic = WaitWest;
    }
}

void TransitionWestToSouth()
{
    PORTB = WestTrafficLight.Yellow;
    PORTB += SouthTrafficLight.Red;
    PORTB += SouthTrafficLight.Yellow;
    wait_milliseconds(oneSec);
    Traffic = GoSouth;
}

#endif /* TRAFFICLIGHTS_H */
```

4 Laboration 4: Objektorienterad programmering i C

4.1 Objektorienterad modellering av en mönstergenerator

I denna uppgift skulle en mönster-visare konstrueras. Programmering skedde i C i en objektorienterad stil. De filer som användes som inkluderingsfiler är `pattern.h` och `pulse.h`. Därefter finns koderna för uppbyggnaden, `pattern.c` och `main.c`. `Pulse.h` har en funktion som läser av negativa flanker och ger en pulserande effekt. De mönster som skapades var följande: En ringräknare, en johnsonräknare, en binär nedräknare och en skiftande blinkmönster.

Algorithm 34 4.1 - `Pulse.h`

```
typedef struct
{
    char in;
    char in_old;
    char out;
} Pulse;

char pulse_run(Pulse* p, char in)
{
    p->in_old = p->in;
    p->in = in;
    p->out = p->in_old & !p->in;
    return p->out;
}
```

Algorithm 35 4.1 - pattern8.h

```
#ifndef PATTERN8_H_
#define PATTERN8_H_
#include "avr/io.h"

enum Pattern {
    Ring,
    Johnson,
    Counter,
    Flash
} patternState;

typedef struct {
    int i;
    int johnson;
    int counter;
} Pattern8;

void pattern8_init(Pattern8* p);
void pattern8_next(Pattern8* p);
void pattern8_run(Pattern8* p);

char pattern8_ring(Pattern8* p);
int pattern8_johnson(Pattern8* p);
int pattern8_counter(Pattern8* p);
int pattern8_flashing(Pattern8* p);

#define pattern (*(volatile Pattern8)* &PORTB)

#endif /* PATTERN8_H_ */
```

Algorithm 36 4.1 - pattern8.c

```
#include "pattern8.h"

pattern8_init(Pattern8* p)
{
    p->i = 0;
    p->johnson = 0;
    p->counter = 0;
}

pattern8_next(Pattern8* p)
{
    pattern8_init(&p);
    switch(patternState)
    {
        case Ring:
            patternState = Johnson;
            break;
        case Johnson:
            patternState = Counter;
            break;
        case Counter:
            patternState = Flash;
            break;
        case Flash:
            patternState = Ring;
            break;
    }
}

char pattern8_ring(Pattern8* p)
{
    char leds[8] = {0x01, 0x02, 0x04, 0x08,
                   0x10, 0x20, 0x40, 0x80};

    p->i++;
    p->i = p->i & 0x07;
    return ~leds[p->i];
}

int pattern8_johnson(Pattern8* p)
{
    if((p->johnson & 0x80) == 0)
        p->johnson=(p->johnson << 1) + 1;
    else
        p->johnson = p->johnson << 1;
    return p->johnson;
}

int pattern8_counter(Pattern8* p)
{
    p->i++;
    return p->i;
}

int pattern8_flashing(Pattern8* p)
{
    if(p->i < 255)
        p->i = p->i << 2;
    else if(p->i > 255)
        p->i = p->i >> 4;
    return p->i;
}
```

Algorithm 37 4.1 - main.c

```
#include <avr/io.h>
#include "pattern8.h"
#include "pulse.h"

void wait_milliseconds(int);

int main()
{
    DDRA = 0x00;
    DDRB = 0xFF;
    Pulse p7;
    Pattern8 p8;

    while(1)
    {
        char sw7 = (!(PINA & 0x80));
        switch(patternState)
        {
            case Ring:
            {
                PORTB = pattern8_ring(&p8);
            } break;
            case Johnson:
            {
                PORTB = pattern8_johnson(&p8);
            } break;
            case Counter:
            {
                PORTB = pattern8_counter(&p8);
            } break;
            case Flash:
            {
                PORTB = pattern8_flashing(&p8);
            }
            default:
            {
                break;
            }
        } // switch

        // Move to next pattern
        if(pulse_run(&p7, sw7))
        {
            pattern8_next(&p8);
        }
        wait_milliseconds(100);
    }
}
```

4.2 Kast av tärningar

4.2.1 main-program

Algorithm 38 4.2 - main.c

```
#include <stdio.h>

#include "Dice.h"
#include "Dice_LCD.h"
#include "Dice_LEDS.h"
#include "UsedLibraries\io_di.h"

int main(void)
{
    io_di button;

    Dice dice1, dice2;
    DiceLCD diceLcd;
    DiceLEDS diceLeds = {0x08, 0x82, 0x52, 0xC6, 0xD6, 0xEE};

    io_di_init(&button, &PORTA, 0x01);

    dice_init(&dice1);
    dice_init(&dice2);
    diceLEDS_init(&diceLeds);
    DiceLCD_init(&diceLcd);

    while (1)
    {
        char ioButton = io_di_read(&button);

        if (!ioButton)
        {
            dice_throw(&dice1);
            dice_throw(&dice2);

            diceLEDS_display(&diceLeds, &dice1);
            DiceLCD_display(&diceLcd, &dice1, &dice2);
        }
    }
    return 0;
}
```

4.2.2 Dice-struct

Algorithm 39 4.2 - Dice.h

```
#ifndef DICE_H_
#define DICE_H_

#include <stdlib.h>

typedef struct {
    int value;
} Dice;

void dice_init(Dice*);
int dice_value(Dice*);
void dice_throw(Dice*);

#endif /* DICE_H_ */
```

Algorithm 40 4.2 - Dice.c

```
#include "Dice.h"

#define DICE_MAX 6

void dice_init(Dice* d)
{
    d->value = 0;
}

void dice_throw(Dice* d)
{
    d->value = rand() % DICE_MAX + 1;
}

int dice_value(Dice* d)
{
    return d->value;
}
```

4.2.3 Dice_LCD-struct

Algorithm 41 4.2 - Dice_LCD.h

```
#ifndef DICE_LCD_H_
#define DICE_LCD_H_

#include <util/delay.h>
#include "UsedLibraries\lcd4.h"
#include "Dice.h"

typedef struct
{
    char s[20];
    lcd4 lcdDisplay;
} DiceLCD;

void diceLCD_init(DiceLCD*);
void diceLCD_display(DiceLCD*, Dice*, Dice*);

#endif /* DICE_LCD_H_ */
```

Algorithm 42 4.2 - Dice_LCD.c

```
#include "Dice_LCD.h"

void DiceLCD_init(DiceLCD* dlcd)
{
    lcd4_init(&dlcd->lcdDisplay, &PORTB, &DDRB, 4000, 100);
}

void DiceLCD_display(DiceLCD* dlcd, Dice* d1, Dice* d2)
{
    sprintf(dlcd->s, "Dice1 = %d", dice_value(d1));
    lcd4_cup_row1(&dlcd->lcdDisplay);
    lcd4_write_string(&dlcd->lcdDisplay, dlcd->s);

    sprintf(dlcd->s, "Dice2 = %d", dice_value(d2));
    lcd4_cup_row2(&dlcd->lcdDisplay);
    lcd4_write_string(&dlcd->lcdDisplay, dlcd->s);
}
```

4.2.4 Dice_LEDs-struct

Algorithm 43 4.2 - Dice_LEDs.h

```
#ifndef DICE_LEDS_H_
#define DICE_LEDS_H_

#include <avr/io.h>
#include "Dice.h"

typedef struct
{
    char leds[6];
} DiceLEDS;

void diceLEDS_init(DiceLEDS*);
void diceLEDS_display(DiceLEDS*, Dice*);

#endif /* DICE_LEDS_H_ */
```

Algorithm 44 4.2 - Dice_LEDs.c

```
#include "Dice_LEDs.h"

diceLEDS_init(DiceLEDS* dleds)
{
    DDRD = 0xFF;
}

void diceLEDS_display(DiceLEDS* dleds, Dice* d)
{
    PORTD = 255 - dleds->leds[d->value - 1];
}
```

4.3 Avläsning av spänningen från en potentiometer

4.3.1 Main-program

Algorithm 45 4.3 - main.c

```
#include <avr/io.h>
#include <stdio.h>
#include "VoltLCD.h"
#include "Potentiometer.h"
#include "io_ai.h"

#define FIVE_VOLT_TIME 1000

int main(void)
{
    DDRD = 0xFF;
    io_ai InVolt;
    io_ai_init(&InVolt, IO_AI_CHANNEL0, 0);

    VoltLCD vLcd;
    Potentiometer potentiometer;

    voltLCD_init(&vLcd);
    potentiometer_init(&potentiometer);

    while (1)
    {
        potentiometer_setvolts(&potentiometer, io_ai_read(&InVolt));
        voltLCD_display(&vLcd, potentiometer_getvolts(&potentiometer));
    }
}
```

4.3.2 Potentiometer-struct

Algorithm 46 4.3 - Potentiometer.h

```
#ifndef POTENTIOMETER_H_
#define POTENTIOMETER_H_

typedef struct
{
    int volts;
} Potentiometer;

void potentiometer_init(Potentiometer*);
int potentiometer_getvolts(Potentiometer*);
void potentiometer_setvolts(Potentiometer*, int);

#endif /* POTENTIOMETER_H_ */
```

Algorithm 47 4.3 - Potentiometer.c

```
#include "Potentiometer.h"

void potentiometer_init(Potentiometer* pm)
{
    pm->volts = 0;
}

int potentiometer_getvolts(Potentiometer* pm)
{
    return pm->volts;
}

void potentiometer_setvolts(Potentiometer* pm, int volts)
{
    pm->volts = volts / 200;
}
```

4.3.3 LCD-struct

Algorithm 48 4.3 - VoltLCD.h

```
#ifndef VOLT_LCD_H_
#define VOLT_LCD_H_

#include <util/delay.h>
#include "lcd4.h"

typedef struct
{
    char s[20];
    lcd4 lcdDisplay;
} VoltLCD;

void voltLCD_init(VoltLCD*);
void voltLCD_display(VoltLCD*, long);

#endif /* VOLT_LCD_H_ */
```

Algorithm 49 4.3 - VoltLCD.c

```
#include "VoltLCD.h"

void voltLCD_init(VoltLCD* vlcd)
{
    lcd4_init(&vlcd->lcdDisplay, &PORTB, &DDRB, 4000, 100);
}

void voltLCD_display(VoltLCD* vlcd, long volts)
{
    sprintf(vlcd->s, "U = %d ", volts);
    lcd4_cup_row1(&vlcd->lcdDisplay);
    lcd4_write_string(&vlcd->lcdDisplay, vlcd->s);
}
```

4.4 Styrning av ringräknarens blinkhastighet med hjälp av en potentiometer

Enda skillnaden med denna uppgift gentemot det föregående är main.c-filen som genom en påbyggnad har fått en funktion som är en ringräknare. Header-filerna som används är också från föregående uppgift.

Algorithm 50 4.4 - main.c

```
#include <avr/io.h>
#include <stdio.h>
#include "VoltLCD.h"
#include "Potentiometer.h"
#include "io_ai.h"

#define FIVE_VOLT_TIME 1000

int RingCounter(int, char*, Potentiometer);

int main(void)
{
    DDRD = 0xFF;
    io_ai InVolt;
    io_ai_init(&InVolt, IO_AI_CHANNEL0, 0);

    VoltLCD vLcd;
    Potentiometer potentiometer;

    voltLCD_init(&vLcd);
    potentiometer_init(&potentiometer);

    char leds[8] =
    {0x01, 0x02, 0x04, 0x08,
     0x10, 0x20, 0x40, 0x80};
    int i = 0;

    while (1)
    {
        potentiometer_setvolts(&potentiometer, io_ai_read(&InVolt));
        voltLCD_display(&vLcd, potentiometer_getvolts(&potentiometer));

        i = RingCounter(i, leds, potentiometer);
    }
}

int RingCounter(int i, char* leds, Potentiometer potentiometer)
{
    i++;
    i = i & 0x07;
    PORTD = ~leds[i];

    int time = FIVE_VOLT_TIME - ←
        (potentiometer_getvolts(&potentiometer) * 200);
    time = time > 0 ? wait_milliseconds(time) : wait_milliseconds(1);
    return i;
}
```

4.5 Temperatursensor SMT160 med PWM-utgång

4.5.1 main-program

Algorithm 51 4.5 - main.c

```
#include <avr/io.h>
#include "lcd4.h"
#include "sensorSMT160.h"
#include "SMTLCD.h"

#define BIT7 0x80

int main(void)
{
    SMTLCD theDisplay;
    SensorSMT160 theTemperature;

    SMTLCD_init(&theDisplay);
    sensorSMT160_init(&theTemperature, &PINB, BIT7);

    while (1)
    {
        sensorSMT160_read(&theTemperature);
        SMTLCD_display(&theDisplay, &theTemperature);
    }
}
```

4.5.2 Sensor SMT160-struct

Algorithm 52 4.5 - SensorSMT160.h

```
#ifndef SENSORSMT160_H_
#define SENSORSMT160_H_

#include <avr/io.h>

typedef struct
{
    int value;
    unsigned char* pin;
    unsigned char bit_mask;
} SensorSMT160;

void sensorSMT160_init(SensorSMT160*, char*, char);
float sensorSMT160_read(SensorSMT160*);
unsigned int sensor_smt160_sample(volatile unsigned char* pin,
                                  unsigned char bit_mask,
                                  unsigned int);

void sensorSMT160_testSignalChange();
void sensorSMT160_testSignalFrequency();

#endif /* SENSORSMT160_H_ */
```

Algorithm 53 4.5 - SensorSMT160.c

```
#include "sensorSMT160.h"

void sensorSMT160_init(SensorSMT160* smt, char* pin, char bit_mask)
{
    smt->pin = pin;
    smt->bit_mask = bit_mask;
    smt->value = 0;
}

float sensorSMT160_read(SensorSMT160* smt)
{
    unsigned int n = 50000;
    unsigned int ones = sensor_smt160_sample(smt->pin, smt->bit_mask, ←
n);
    float dc = (float) ones / (float) n;
    float temp = (dc - 0.32) / 0.00470;

    smt->value = temp;
    return temp;
}

__attribute__((noinline))
unsigned int sensor_smt160_sample(
volatile unsigned char* pin, //Register R25:R24
unsigned char bit_mask, //Register R22
unsigned int n //Register R21:R20
)
{
    {
        asm volatile (
            " movw r30, r24 " "\n\t"
            " movw r26, r20 " "\n\t"
            " ldi r24, 0x00 " "\n\t"
            " ldi r25, 0x00 " "\n\t"
            " ldi r19, 0xff " "\n\t"
            " ldi r20, 0x00 " "\n\t"
            "loop: " "\n\t"
            " ld r18, Z " "\n\t"
            " and r18, r22 " "\n\t"
            " add r18, r19 " "\n\t"
            " adc r24, r20 " "\n\t"
            " adc r25, r20 " "\n\t"
            " sbiw r26, 0x01 " "\n\t"
            " brne loop " "\n\t"
            " ret " "\n\t"
        );
    }
}

void sensorSMT160_testSignalChange()
{
    /* Test that there's a sensorsignal on the digital input.
    If signal is constant for a period of time ie 10 ms,
    then its something wrong on the sensor */
}

void sensorSMT160_testSignalFrequency()
{
    /* The frequency on the signal from the sensor shall be in the
    interval of 1-4 kHz. If it's not then there's probably sensorfault.
    Tests if the frequency on the sensorsignal is in a given ←
interval. */
}
```

4.5.3 LCD-struct

Algorithm 54 4.5 - SMTLcd.h

```
#ifndef SMTLCD_H_
#define SMTLCD_H_

#include <util/delay.h>
#include "lcd4.h"
#include "sensorSMT160.h"

typedef struct
{
    char s[20];
    lcd4 lcdDisplay;
} SMTLCD;

void SMTLCD_init(SMTLCD*);
void SMTLCD_display(SMTLCD*, SensorSMT160*);

#endif /* SMTLCD_H_ */
```

Algorithm 55 4.5 - SMTLcd.c

```
#include "sensorSMT160.h"
#include "SMTLCD.h"

void SMTLCD_init(SMTLCD* smtLcd)
{
    lcd4_init(&smtLcd->lcdDisplay, &PORTB, &DDRB, 4000, 100);
}

void SMTLCD_display(SMTLCD* smtLcd, SensorSMT160* smt)
{
    sprintf(smtLcd->s, "Temperature:");
    lcd4_cup_row1(&smtLcd->lcdDisplay);
    lcd4_write_string(&smtLcd->lcdDisplay, smtLcd->s);

    sprintf(smtLcd->s, "%d", smt->value);
    lcd4_cup_row2(&smtLcd->lcdDisplay);
    lcd4_write_string(&smtLcd->lcdDisplay, smtLcd->s);

    lcd4_write_char(&smtLcd->lcdDisplay, 0xDF);
    lcd4_write_char(&smtLcd->lcdDisplay, 'C');
}
```


4.6 Öka/Minska-knappar med acceleration

4.6.1 Main-program

Algorithm 56 4.6 - main.c

```
#include <avr/io.h>
#include "Accelerator.h"
#include "AccLCD.h"

char increasebutton_read();
char decreasebutton_read();

int main(void)
{
    AccLCD display;
    Accelerator acc;

    AccLCD_init(&display);
    accelerator_init(&acc);

    while (1)
    {
        char increaseButton = increasebutton_read();
        char decreaseButton = decreasebutton_read();
        switch (countingState)
        {
            case IDLE:
            {
                accelerator_idle(&acc, increaseButton, decreaseButton);
            } break;
            case INCREMENT:
            {
                accelerator_increment(&acc, increaseButton);
            } break;
            case DECREASE:
            {
                accelerator_decrease(&acc, decreaseButton);
            } break;
        }
        AccLCD_display(&display, &acc);
    }
}

char increasebutton_read()
{
    return (!(PINA & 0x01));
}

char decreasebutton_read()
{
    return (!(PINA & 0x02));
}
```

4.6.2 Accelerator-struct

Algorithm 57 4.6 - Accelerator.h

```
#ifndef ACCELERATOR_H_
#define ACCELERATOR_H_
#include <avr/io.h>

enum Counting {
    INCREMENT,
    DECREASE,
    IDLE
} countingState;

typedef struct
{
    int value;
    int counter;
    int accelerationStart;
    unsigned char button;
} Accelerator;

void accelerator_init(Accelerator*);
void accelerator_increment(Accelerator*, char);
void accelerator_decrease(Accelerator*, char);
void accelerator_delay(Accelerator*, char);
void accelerator_idle(Accelerator*, char, char);

#endif /* ACCELERATOR_H_ */
```

Algorithm 58 4.6 - Accelerator.c

```
#include "Accelerator.h"

wait_milliseconds(int);

void accelerator_init(Accelerator* acc)
{
    DDRA = 0x00;
    countingState = IDLE;
    acc->value = 0;
    acc->counter = 0;
    acc->accelerationStart = 2;
}

void accelerator_increment(Accelerator* acc, char button)
{
    acc->value++;
    accelerator_delay(acc, button);
}

void accelerator_decrease(Accelerator* acc, char button)
{
    acc->value--;
    accelerator_delay(acc, button);
}

void accelerator_delay(Accelerator* acc, char button)
{
    if(button && acc->counter > acc->accelerationStart)
    {
        wait_milliseconds(100);
    }
    else if(button && acc->counter <= acc->accelerationStart)
    {
        wait_milliseconds(1000);
    }
    else
    {
        countingState = IDLE;
    }
    acc->counter++;
}

void accelerator_idle(Accelerator* acc,
char increaseButton, char decreaseButton)
{
    if(increaseButton)
    {
        countingState = INCREMENT;
    }
    else if(decreaseButton)
    {
        countingState = DECREASE;
    }
    acc->counter = 0;
}
```

4.6.3 LCD-struct

Algorithm 59 4.6 - AccLCD.h

```
#ifndef ACCLCD_H_
#define ACCLCD_H_

#include <util/delay.h>
#include "lcd4.h"
#include "Accelerator.h"

typedef struct
{
    char s[20];
    lcd4 lcdDisplay;
} AccLCD;

void AccLCD_init(AccLCD*);
void AccLCD_display(AccLCD*, Accelerator*);

#endif /* ACCLCD_H_ */
```

Algorithm 60 4.6 - AccLCD.c

```
#include "AccLCD.h"

void AccLCD_init(AccLCD* accLcd)
{
    lcd4_init(&accLcd->lcdDisplay, &PORTB, &DDRB, 4000, 100);
}

void AccLCD_display(AccLCD* accLcd, Accelerator* acc)
{
    sprintf(accLcd->s, "Value = %d", acc->value);
    lcd4_cup_row1(&accLcd->lcdDisplay);
    lcd4_write_string(&accLcd->lcdDisplay, accLcd->s);
}
```

4.7 Morse-kodlås

I denna uppgift konstruerades en morse-kodlås som har två tillståndsmaskiner. Den ena sköter om inmatning och den andra sköter om kodlåset.

4.7.1 Main-program

Ursäkta för koden i main, som inte har genomgått någon större refaktorisering. Ett försök att paketera in koderna i sina egna klasser gjordes men resultatet blev inte som förväntat. Eftersom det tar tid att implementera så kommer det att utföras vid ett senare tillfälle.

Just nu innehåller filen båda tillståndsmaskinerna.

Algorithm 61 4.7 - main.c del I

```
#include <avr/io.h>
#include "MorseLCD.h"
#include "Morse.h"
#include "time.h"

void MorseLock();
void MorseInput();
char MorseResetCheck(char, unsigned, const unsigned, Morse);
void wait_milliseconds(int);

int main(void)
{
    MorseLCD morseLcd;
    morseLCD_init(&morseLcd);

    Morse morse;
    morse_init(&morse);
    morse_reset(&morse);

    char code_count = 0;

    unsigned tick_counter = 0;
    const unsigned tick = 10;
    unsigned reset_counter = 0;
    const unsigned reset_morse = 20;

    char const access_code = 5;      // Passcode
    char attempted_codes[6];
    const int lock_timeout = 20;
    int timeout_counter = 0;
    LockState = LOCKED;

    while (1)
    {
        MorseInput();
        MorseLock();

        // Resets LCD-display after 20 ticks
        code_count = MorseResetCheck(code_count, reset_counter,
                                     reset_morse, morse);
    }
}

char MorseResetCheck(
char code_count,
unsigned reset_counter,
const unsigned reset_morse,
Morse morse)
{
    if (code_count == 5 || reset_counter > reset_morse)
    {
        code_count = 0;
        wait_milliseconds(1000);
        morse_reset(&morse);
    }
    return code_count;
}
```

Algorithm 62 4.7 - main.c del II

```
void MorseLock()
{
    switch(LockState)
    {
        case LOCKED:
        {
            morse_value(&morse);
            attempted_codes[code_count] = morse.value;
            if(code_count == 5 &&
               attempted_codes[code_count] == access_code)
            {
                LockState = UNLOCKED;
            }

            morseLCD_displaylock(&morseLcd, &morse, 0);
        } break;

        case UNLOCKED:
        {
            if(timeout_counter > lock_timeout)
            {
                LockState = LOCKED;
                timeout_counter = 0;
                code_count = 0;
            }
            timeout_counter++;
            morseLCD_displaylock(&morseLcd, &morse, 1);
        } break;
    } //switch(LockState)
}

void MorseInput()
{
    char button = (!(PINA & 0x01));
    switch ( ButtonState )
    {
        case IDLE:
        {
            tick_counter = 0;
            reset_counter++;
            if(button)
            {
                ButtonState = PUSHED;
                reset_counter = 0;
            }
        } break;
        case PUSHED:
        {
            tick_counter++;
            if(!button)
            {
                if(tick_counter < tick)
                {
                    morse_short(&morse, code_count);
                }
                if(tick_counter > tick)
                {
                    morse_long(&morse, code_count);
                }
                code_count++;
                ButtonState = IDLE;
            }
        } break;
    } // switch(ButtonState)
}
```

4.7.2 Morse-struct

Algorithm 63 4.7 - Morse.h

```
#ifndef MORSE_H_
#define MORSE_H_

#include <avr/io.h>

enum
{
    IDLE,
    PUSHED
} ButtonState;

enum
{
    UNLOCKED,
    LOCKED
} LockState;

typedef struct
{
    int value;
    char code[6];
} Morse;

void morse_init(Morse* m);
void morse_reset(Morse* m);
void morse_long(Morse* m, int);
void morse_short(Morse* m, int);
void morse_value(Morse* m);

#endif /* MORSE_H_ */
```

Algorithm 64 4.7 - Morse.c

```
#include "Morse.h"

const char morse_codes[10][6] =
{
    "_____", //0
    "._____", //1
    ".._____", //2
    "..._____", //3
    "...._____", //4
    ".....", //5
    "____.", //6
    "_____", //7
    "____.", //8
    "_____", //9
};

morse_init(Morse* m)
{
    m->value = 0;
}

void morse_reset(Morse* m)
{
    for (int i=0; i<7; i++)
    {
        m->code[i]=' ';
    }
}

void morse_long(Morse* m, int code_count)
{
    m->code[code_count]='_';
}

void morse_short(Morse* m, int code_count)
{
    m->code[code_count]='.';
}

void morse_value(Morse* m)
{
    volatile int index = 0;

    for (int i = 0; i <= 10; i++)
    {
        m->code[5]=0;
        if(strcmp(morse_codes[i], m->code) == 0)
        {
            m->value = i; break;
        }
    }
}
```


4.7.3 LCD-struct

Algorithm 65 4.7 - MorseLCD.h

```
#ifndef MORSELCD_H_
#define MORSELCD_H_

#include <util/delay.h>
#include "lcd4.h"
#include "Morse.h"

typedef struct
{
    char s[20];
    lcd4 lcdDisplay;
} MorseLCD;

void morseLCD_init(MorseLCD*);
void morseLCD_display(MorseLCD*, Morse* m);

#endif /* MORSELCD_H_ */
```

Algorithm 66 4.7 - MorseLCD.c

```
#include "MorseLCD.h"
#include "Morse.h"

void morseLCD_init(MorseLCD* mlcd)
{
    lcd4_init(&mlcd->lcdDisplay, &PORTB, &DDRB, 700, 50);
}

void morseLCD_display(MorseLCD* mlcd, Morse* m)
{
    m->code[7] = 0;

    sprintf(mlcd->s, "Morse: %s", m->code);
    lcd4_cup_row1(&mlcd->lcdDisplay);
    lcd4_write_string(&mlcd->lcdDisplay, mlcd->s);

    sprintf(mlcd->s, "Value: %d", m->value);
    lcd4_cup_row2(&mlcd->lcdDisplay);
    lcd4_write_string(&mlcd->lcdDisplay, mlcd->s);
}

void morseLCD_displaylock(MorseLCD* mlcd, Morse* m, int locked)
{
    m->code[7] = 0;
    sprintf(mlcd->s, "Morse: %s", m->code);
    lcd4_cup_row1(&mlcd->lcdDisplay);
    lcd4_write_string(&mlcd->lcdDisplay, mlcd->s);

    char *s;
    if(locked)
    {
        s = "UNLOCKED";
    }
    else
    {
        s = "LOCKED ";
    }

    char buf[1024];
    memset(buf, 0, sizeof buf);
    strncpy(buf, s, (sizeof buf)-1);

    sprintf(mlcd->s, "%s", s);
    lcd4_cup_row2(&mlcd->lcdDisplay);
    lcd4_write_string(&mlcd->lcdDisplay, s);
}
```