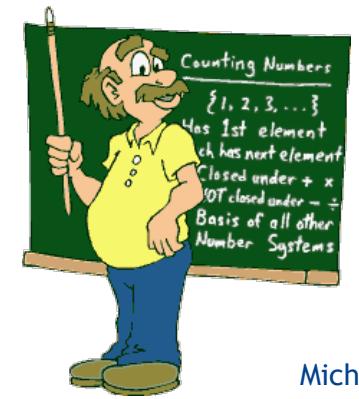


MSYS

Microcontroller Systems

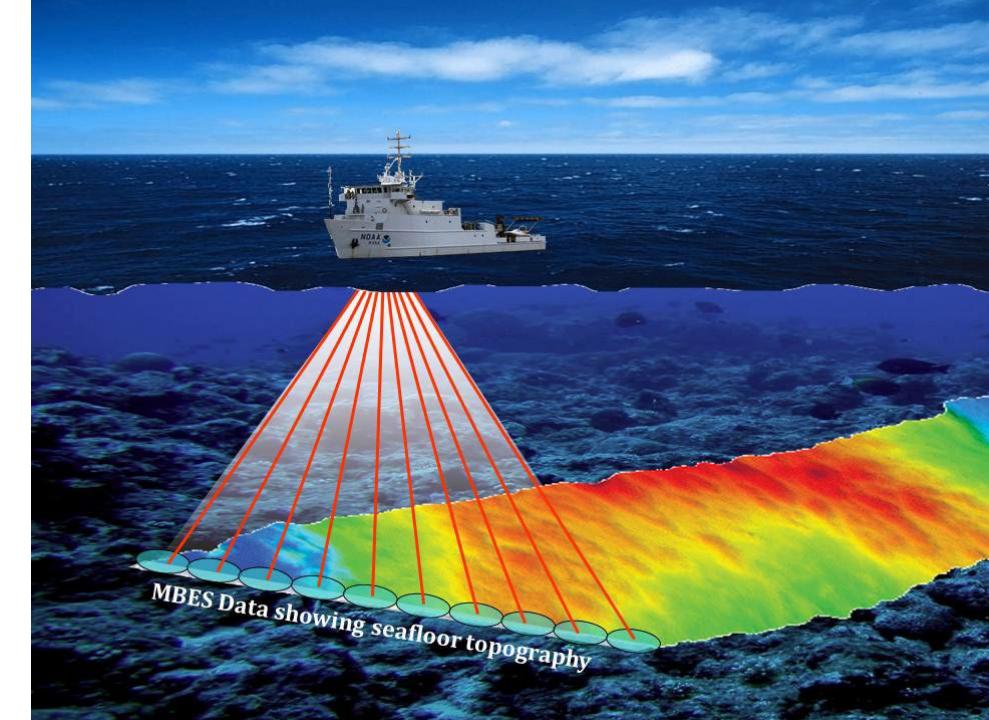
Lektion 1: Introduktion til MSYS



Slides from: Henning Hargaard

Michael Alrøe

Underviser



Michael Alrøe
ma@ece.au.dk

Kontor: 5125-214 (Edison)

<https://dk.linkedin.com/in/alroe>



Hjælpelærere (i LAB)



Lars Mortensen
lamo@ece.au.dk

Studerende

Øvelser

SHANNON BYGNINGEN - FINLANDSGADE 22

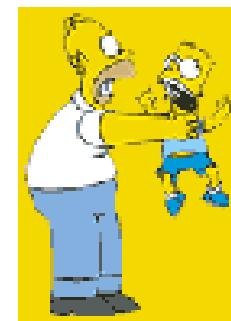
STUEN



Læringsmål

Når kurset er afsluttet, forventes den studerende at kunne:

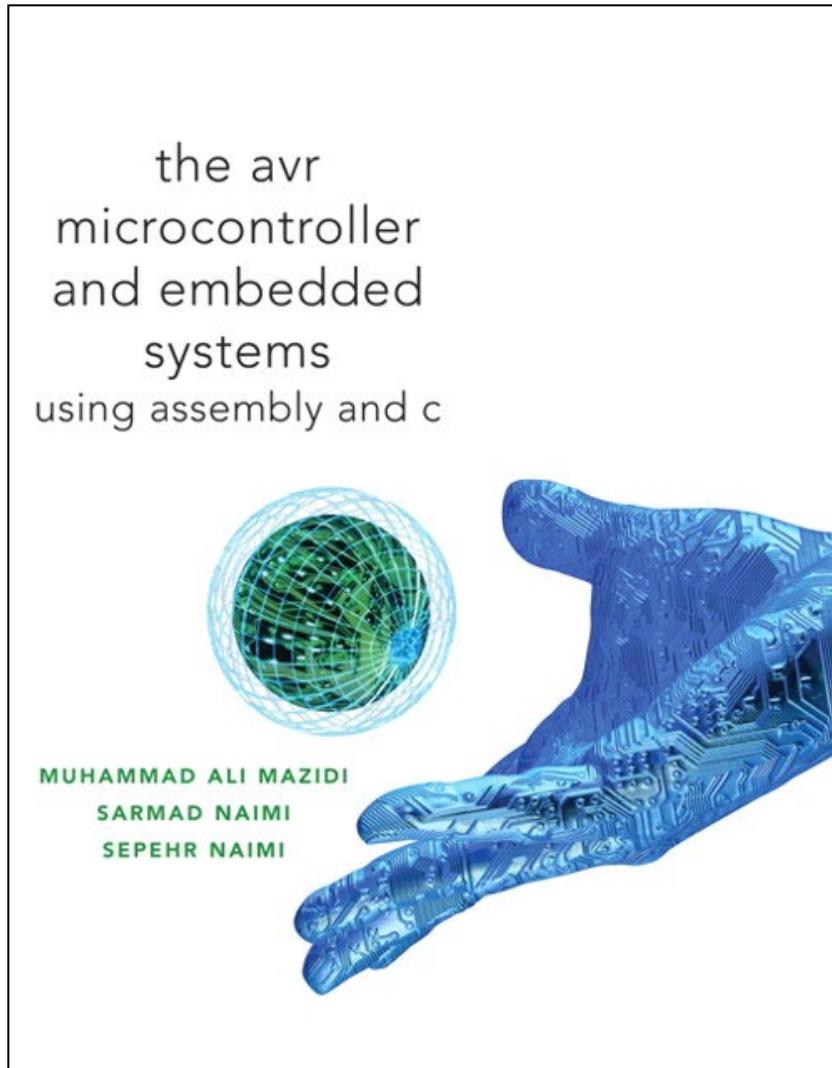
- Beskrive en 8-bit microcontrollers interne arkitektur: CPU, Timer, I/O enheder etc.
 - Anvende assembly og C til programmering af en dedikeret microcontroller
 - Implementere og teste drivere for grundlæggende I/O-enheder (parallelle porte)
 - Implementere og teste drivere for seriell, asynkron kommunikation (UART)
 - Anvende interrupts
 - Anvende hardware-timere
 - Anvende A/D-konvertere
- > Eksamensform: Multiple Choice, karakter efter 7-skalaen.



Indhold

- Computer-arkitektur generelt.
- Atmel AVR microcontrollerens opbygning.
- Mega32 og Mega2560.
- Arduino Mega2560 + Mega2560 I/O shield.
- Assembly-programmering.
- Microchip Studio (assembler + compiler + debugger).
- C programmering (GCC compiler).
- Anvendelse/programmering af timers.
- Anvendelse/programmering af interrupts.
- Seriel kommunikation (UART / COM-porte).
- Anvendelse/programmering af A/D-converters.

Lærebogen



Bogen anvender
Microcontrolleren AVR
Mega32.

I starten programmeres i
assembly.

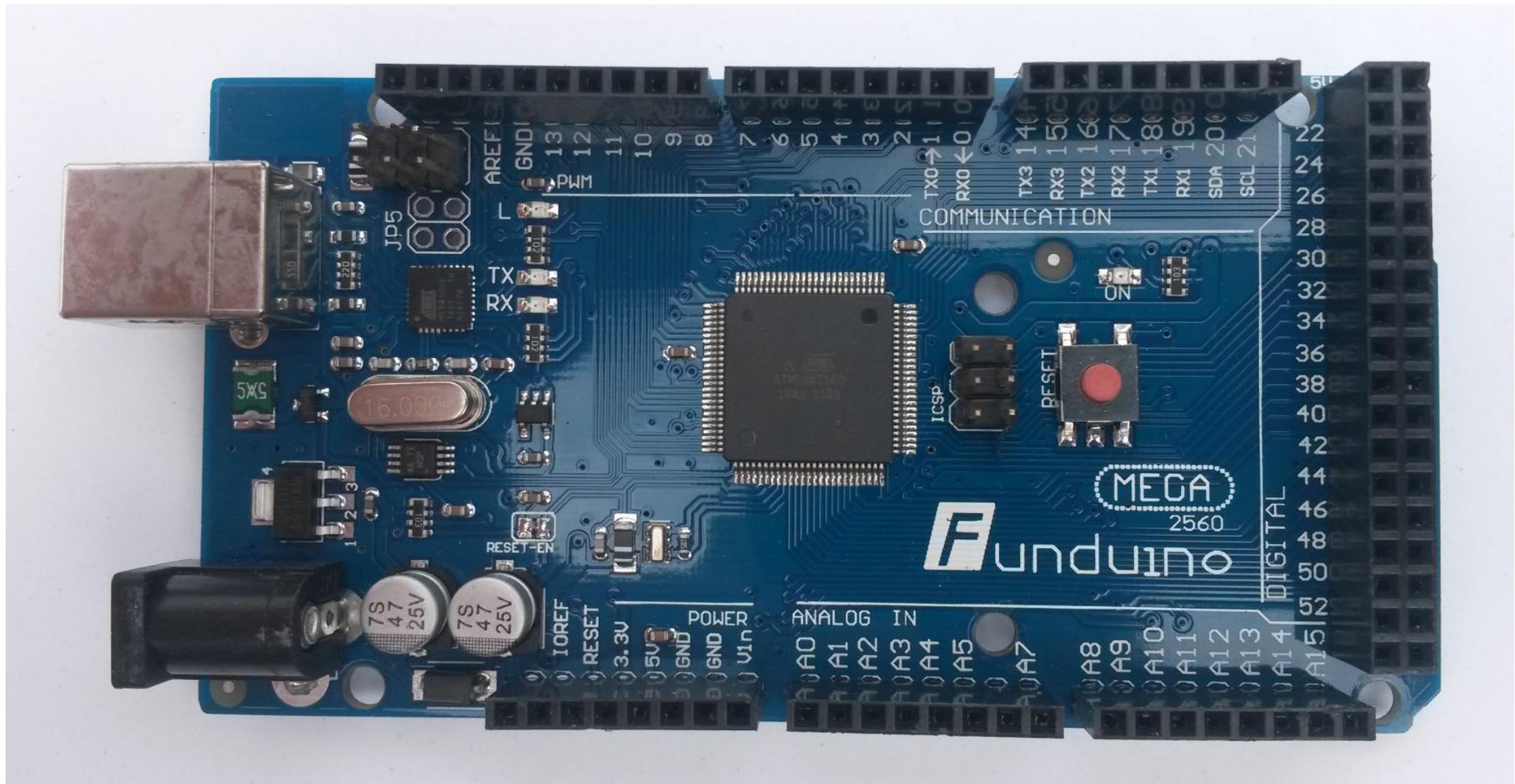
Senere programmeres i
C.

Der læses i gennemsnit
cirka 20 sider **FØR** hver
lektion.

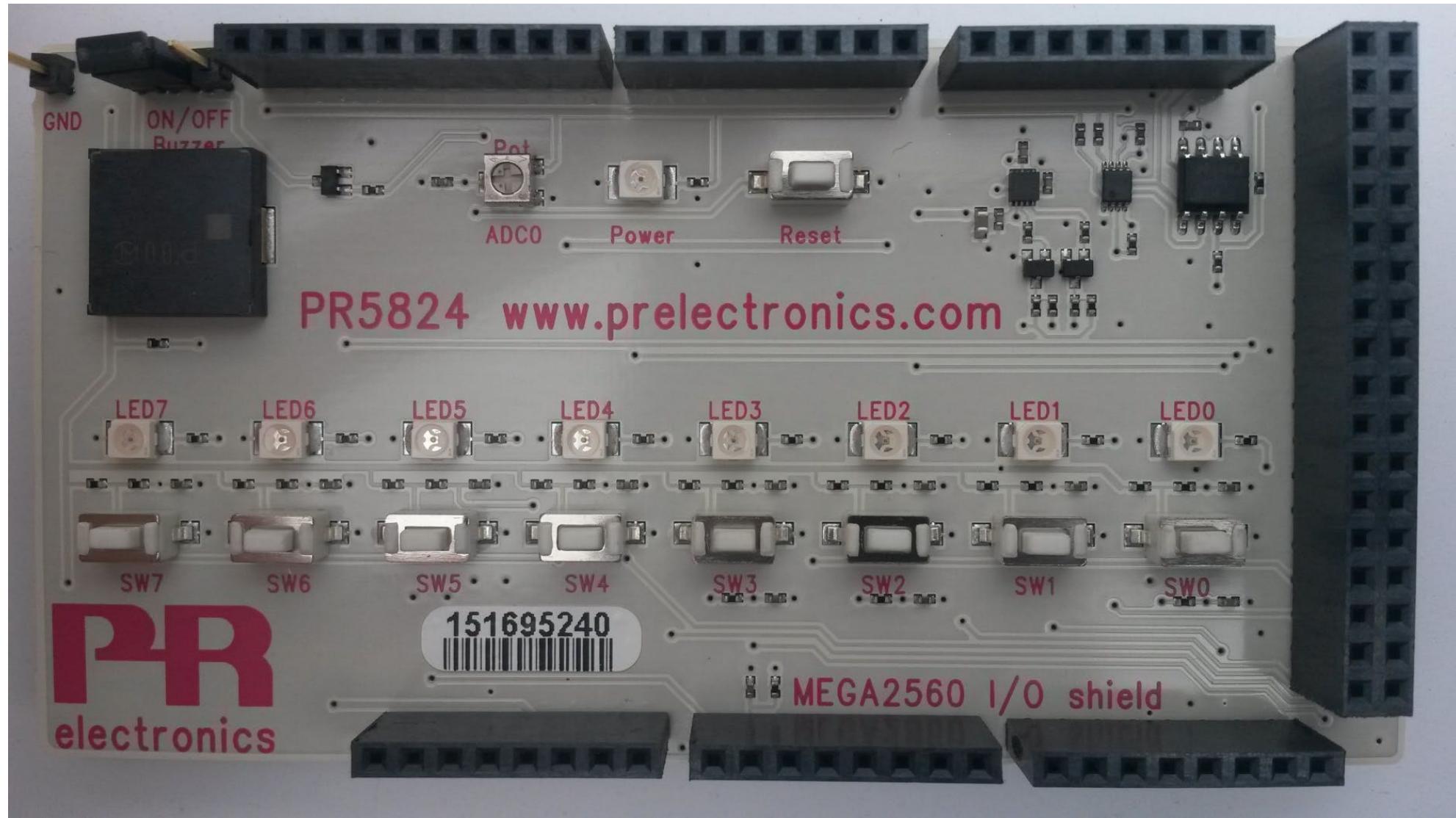
Hardware og bøger

- **Lærebogen:** Mazidi: "The AVR Microcontroller and Embedded System".
- "Arduino Mega2560" + USB kabel (bogladen).
- "Mega2560 I/O Shield" (bogladen).
- Hæftet "Mega2560 I/O Registers" (bogladen).
- **Microchip Studio 7** assembler + debugger + C compiler (gratis, Brightspace).
- Databøger for Mega32 og for Mega2560 (pdf, Brightspace).
- Diverse manualer på Brightspace (pdf, Brightspace).
- Øvelsesvejledninger (pdf, Brightspace).

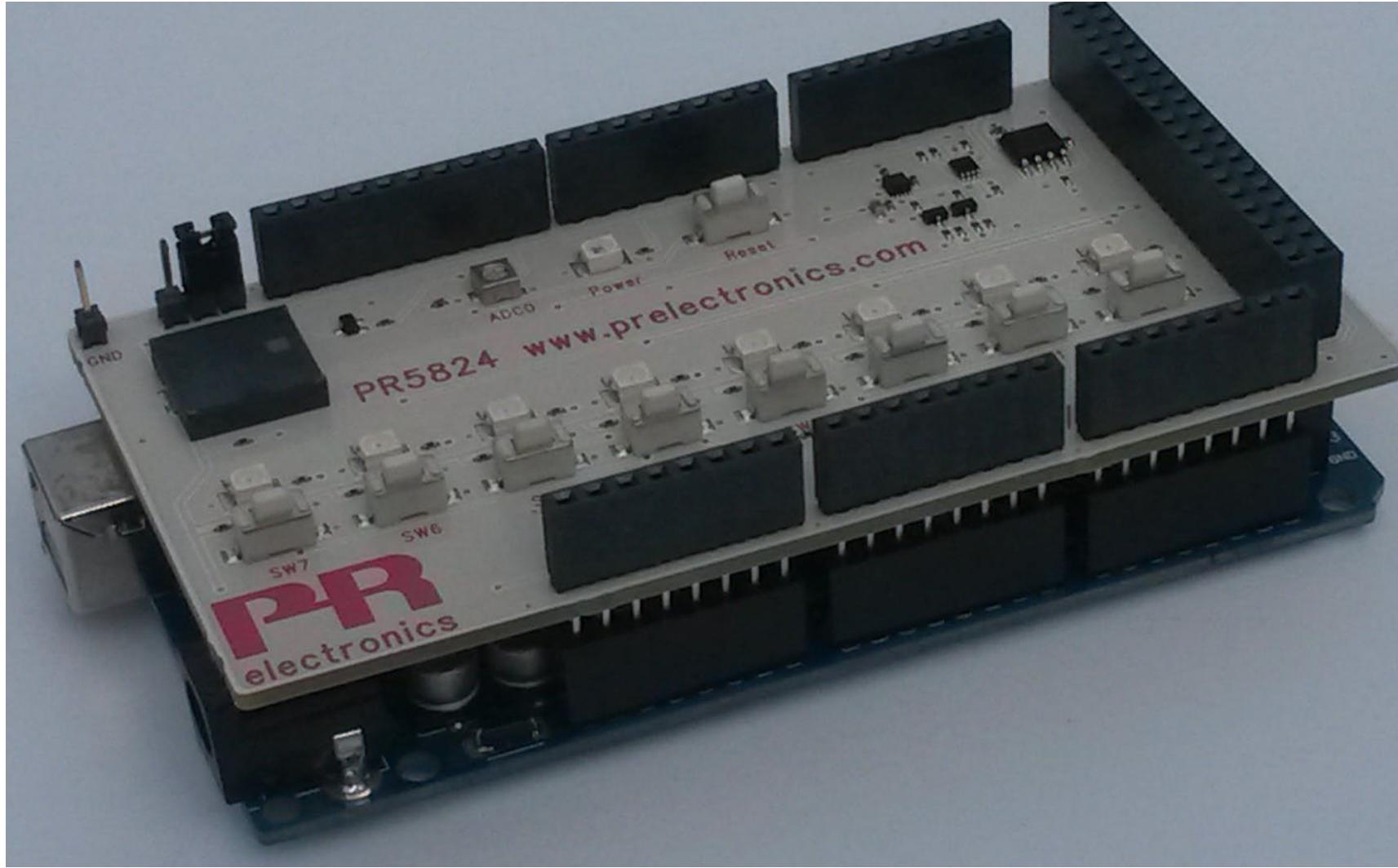
Arduino Mega2560



Mega2560 I/O Shield



Arduino Mega2560 monteret med shield

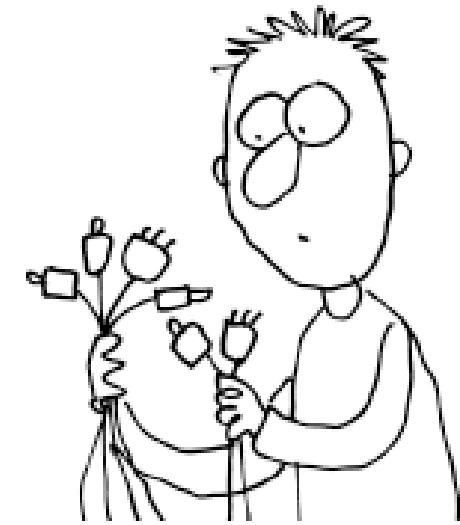


Følg med på Brightspace !

- Meddelelser.
- Læringsstier (Content).

Hvordan får vi styr på alt det ?

- Via lærebogen.
- Via lektioner på klassen.
- **Via øvelser !**



Erfaringen viser, at **"learning by doing"** er en meget effektiv læringsform !

Slut



Spørgsmål / kommentarer ?



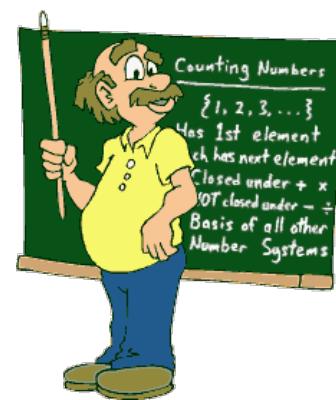
AARHUS
UNIVERSITY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

MSYS

Microcontroller Systems

Lektion 2_1: Introduktion til computere



Slides from: Henning Hargaard

Hvad er en computer ?



- Grundlæggende er en ”computer” blot en elektronisk enhed, der er i stand til at foretage beregninger.
- En PC er en ”general purpose” computer.
- Mange apparater indeholder en ”special purpose” computer (typisk en **microcontroller**).



Hvad er computer-arkitektur ?

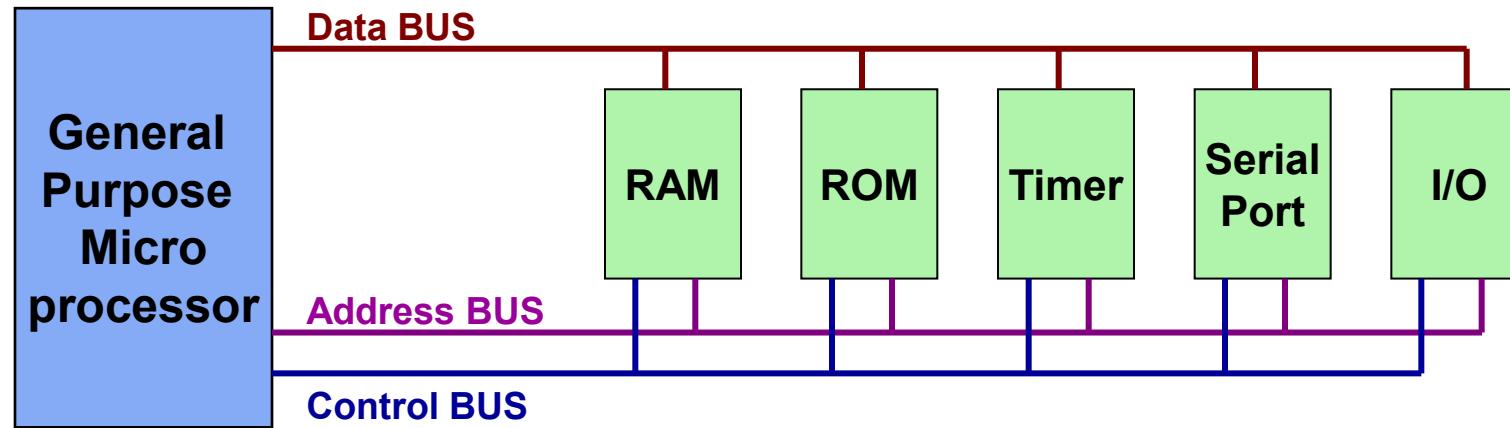
Måden, hvorpå en computer internt er opbygget (HW).

Hvad skal en computer indeholde ?

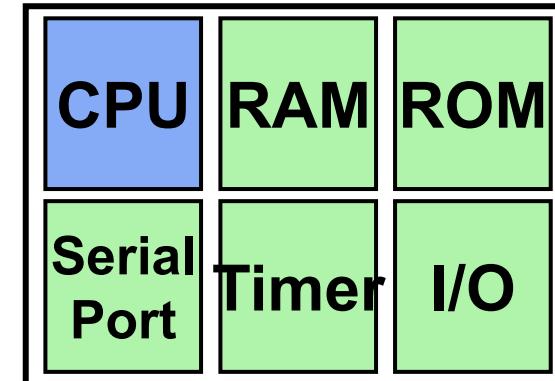
- Regne-enhed (ALU) og status-register.
- Hukommelse (program og data).
- Program-tæller.
- Instruktions-dekoder.
- Input og output – enheder.

General Purpose Microprocessors vs. Microcontrollers

- General Purpose Microprocessors



- Microcontrollers



Microcontroller - eksempler

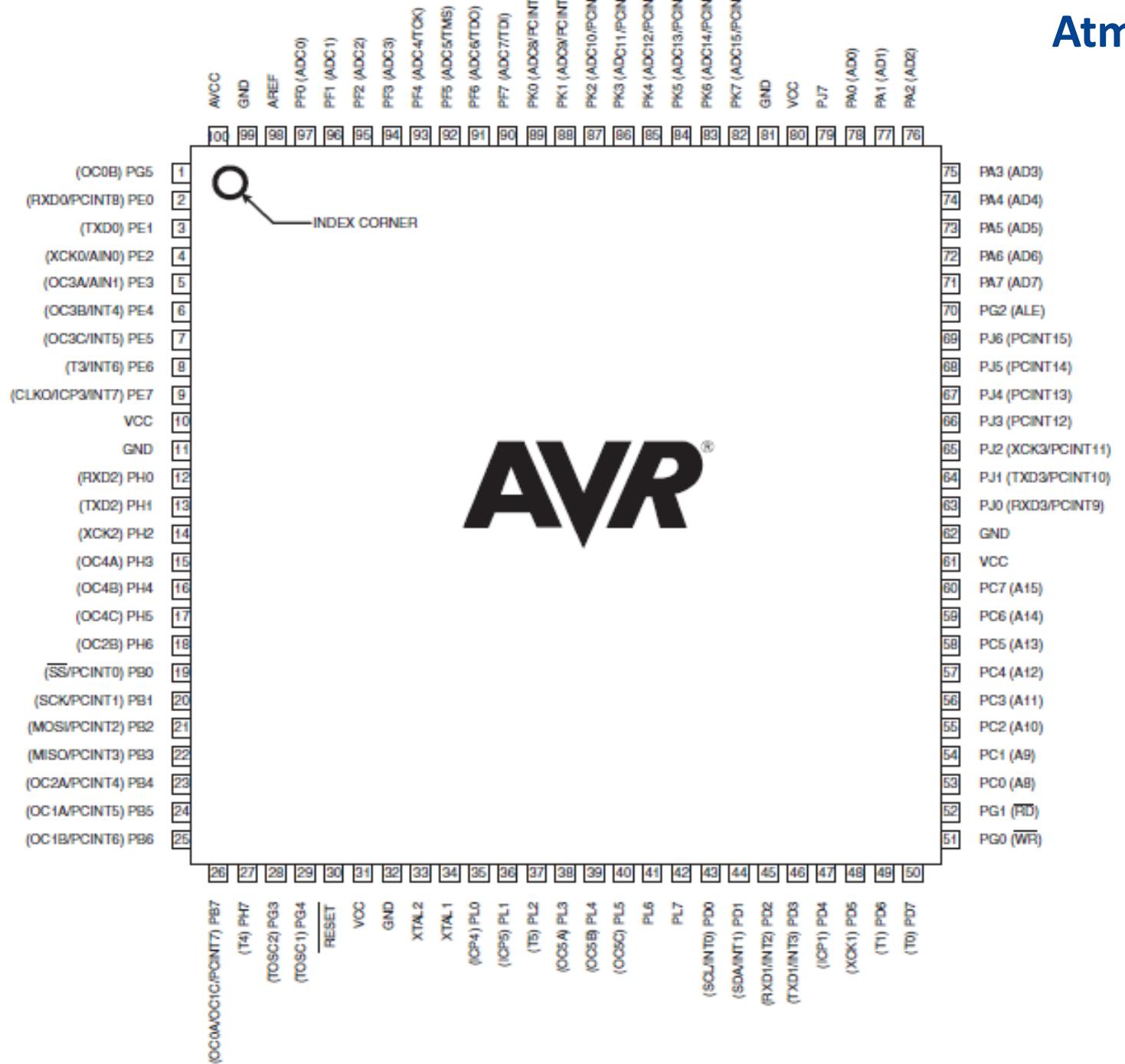


Atmel AVR: Single chip microcontroller

PDIP		
(XCK/T0)	PB0	1
(T1)	PB1	2
(INT2/AIN0)	PB2	3
(OC0/AIN1)	PB3	4
(SS)	PB4	5
(MOSI)	PB5	6
(MISO)	PB6	7
(SCK)	PB7	8
RESET		9
VCC		10
GND		11
XTAL2		12
XTAL1		13
(RXD)	PD0	14
(TXD)	PD1	15
(INT0)	PD2	16
(INT1)	PD3	17
(OC1B)	PD4	18
(OC1A)	PD5	19
(ICP1)	PD6	20
		40
		39
		38
		37
		36
		35
		34
		33
		32
		31
		30
		29
		28
		27
		26
		25
		24
		23
		22
		21
		PA0 (ADC0)
		PA1 (ADC1)
		PA2 (ADC2)
		PA3 (ADC3)
		PA4 (ADC4)
		PA5 (ADC5)
		PA6 (ADC6)
		PA7 (ADC7)
		AREF
		GND
		AVCC
		PC7 (TOSC2)
		PC6 (TOSC1)
		PC5 (TDI)
		PC4 (TDO)
		PC3 (TMS)
		PC2 (TCK)
		PC1 (SDA)
		PC0 (SCL)
		PD7 (OC2)

Pinout for
Mega32
(den fra lærebogen)

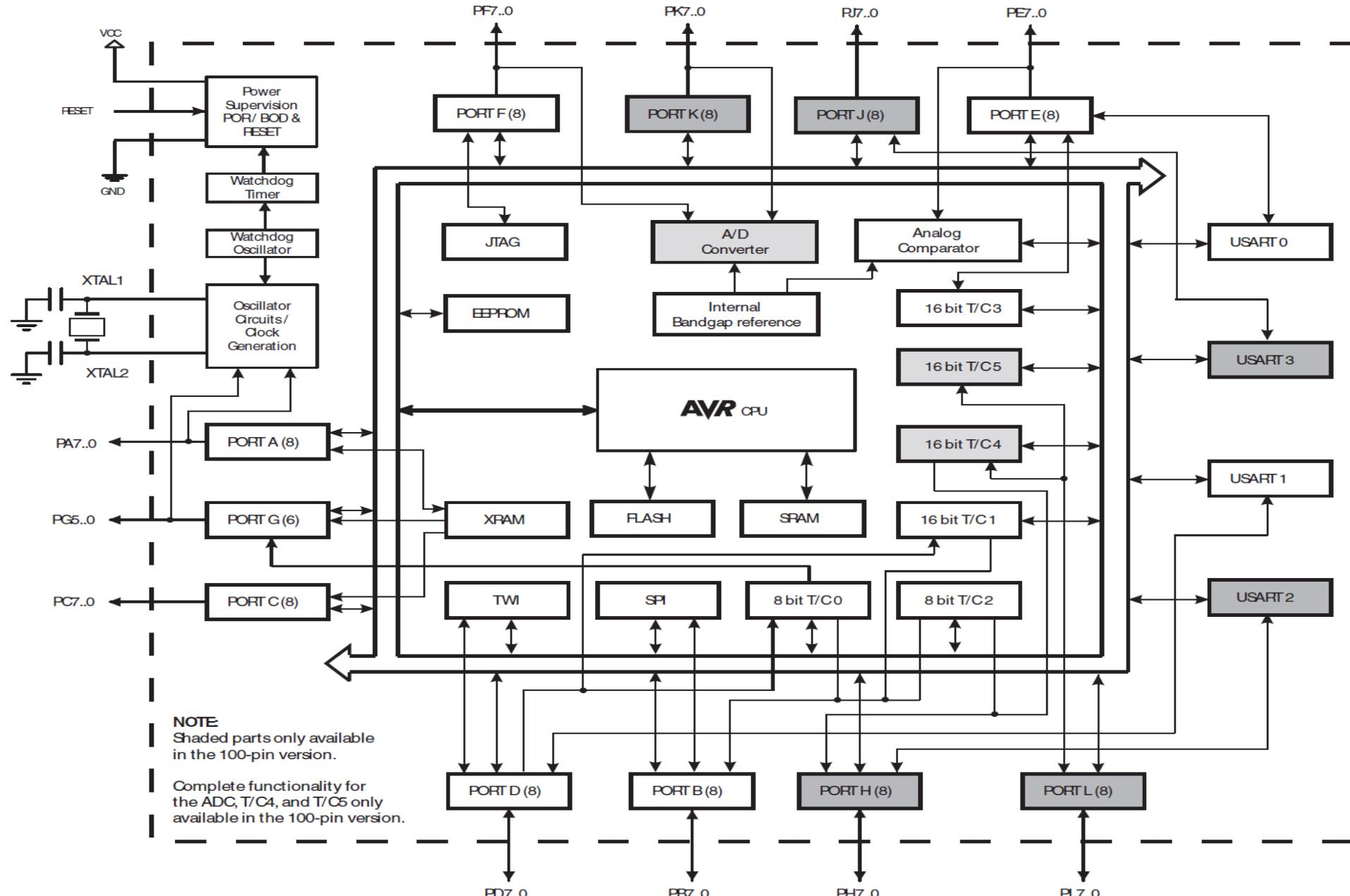
Atmel AVR: Single chip microcontroller



Pinout for Mega2560

Bruges i øvelserne!

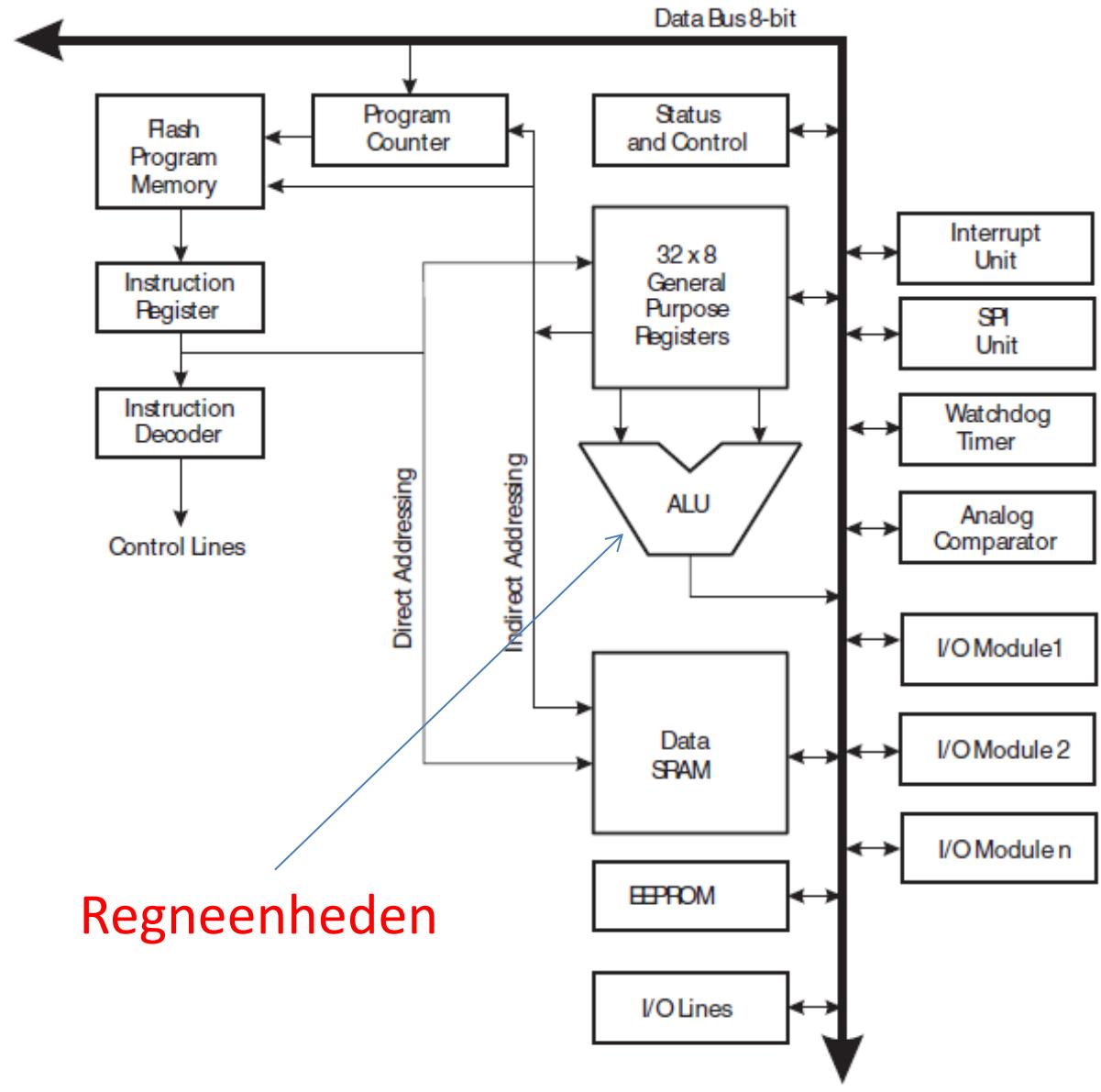
Mega2560: Intern Arkitektur



AVR ATmega arkitektur - generelt

Alle CPU'er har:

- Arithmetic Logic Unit (ALU)
- Program Counter (PC)
- Instruction decoder
- Registers



Program (software)

- Et *program* er den **sekvens af instruktioner**, som beskriver, hvad en computer skal udføre.
- Programmet skal overføres ("**downloades**" eller "**installeres**") på computeren, før det kan afvikles.
- En **PC** udfører typisk mange forskellige programmer til forskellige tidspunkter (afhængig af formålet).
- En **microcontroller** vil typisk styre et apparat med en **bestemt funktionalitet** ("special purpose"), og programmet installeres derfor ofte en gang for alle.

Maskinkode

- Grundlæggende kan en microcontroller **kun** forstå og afvikle maskinkoder, der er **binære talkoder**.
- Koderne afvikles sekventielt ("efter tur"), og styrer på **primitiv** vis microcontrollerens **interne hardware**.
- Man anvender ofte symbolsk maskinkode (**ASSEMBLY**), når programmet skrives.
- Symbolsk maskinkode oversættes til maskinkode med en **assembler** (et program, der kører på en PC).

Eksempel (ASSEMBLY):

```
LDS R26,_led_status  
CPI R26,LOW(0xFF)  
BRNE _0x4
```

Højniveau - sprog

- Det er **besværligt**, og kræver dyb indsigt i computerens interne arkitektur at kunne skrive **assembly**-kode.
- Langt overvejende anvendes derfor programmering på et højere abstraktions-niveau.
- Eksempler er: Pascal, Basic, Java, C++, C, C#.
- Højniveau-sprog skal **oversættes til maskinkoder** for at kunne forstås og afvikles af computeren. Hertil kan anvendes et program, der kaldes en **COMPILER**.

Eksempel:

```
if ( der_er_trykket_paa_knappen() )
    start_motoren();
else
    stop_motoren();
```

C kontra Assembly

- C :

```
if (led_status==0xff)
```

- Assembly :

```
LDS R26,_led_status  
CPI R26,LOW(0xFF)  
BRNE _0x4
```

Et C program

- C er meget anvendt til **programmering af microcontrollers**.

Eksempel:

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

int main()
{
    unsigned char i = 0;
    DDRA = 0xFF; //port A as output
    DDRB = 0xFF; //port B as output
    DDRC = 0xFF; //port C as output
    PORTA = 0xAA;
    while (1)
    {
        PORTC = PORTC ^ 0x01; //toggle PORTC.0
        PORTB = i;
        i++;
    }
    return 0;
}
```

Compiler

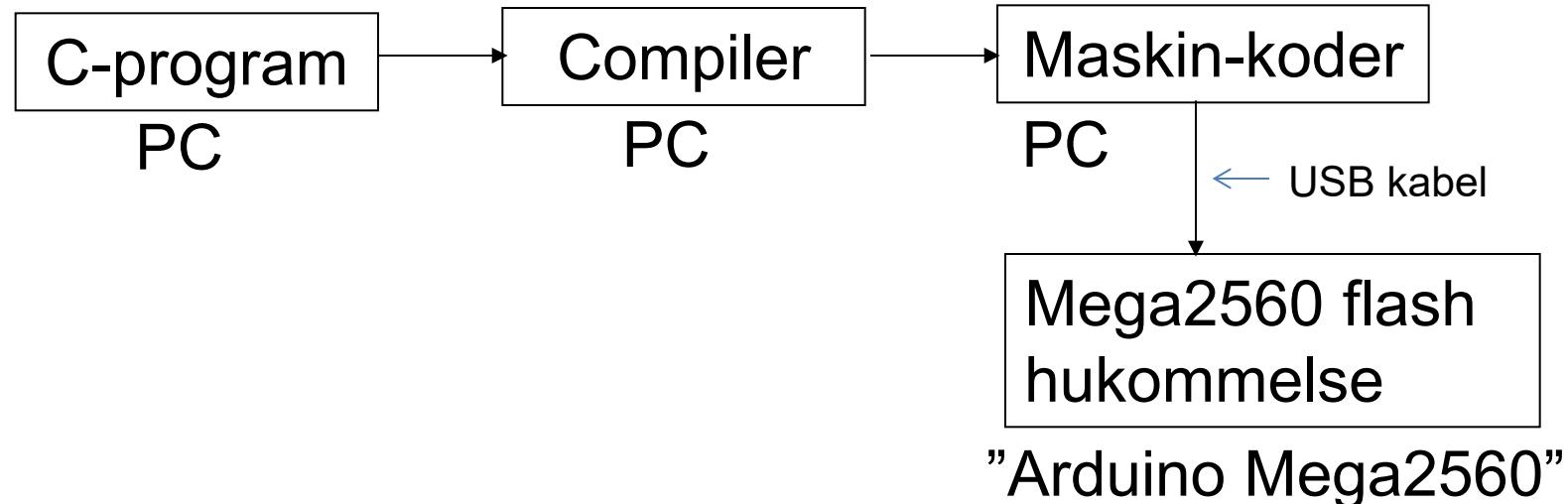
- Compileren ”oversætter” vores **C-kode** til en fil, der indeholder alle **maskinkoderne** for programmet.

Forskellige computere har forskellige maskinkoder =>
individuelle compilere.

- Compileren er **et program**, der typisk kører på en PC.

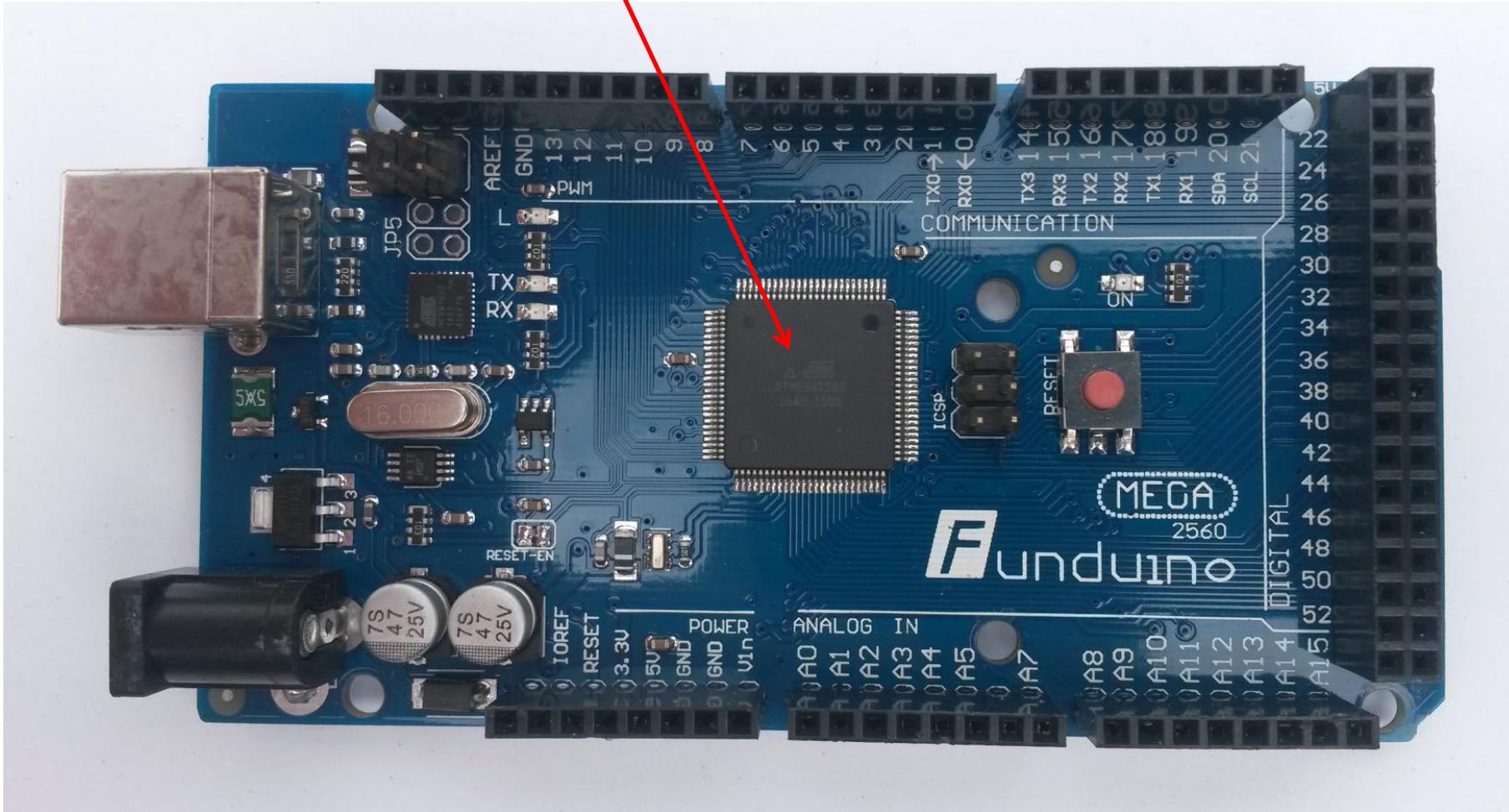
Program - download

- Når C-programmet er oversat af compileren (på PC'en), skal programmet **overføres til "vores computer"** (= target).
- Dette kan foregå simpelt via et USB kabel.
- I LAB er vores "target" microcontrolleren AVR Mega2560. Den har en speciel hukommelse (Flash), der **husker programmet, selv om vi slukker for strømmen.**



"Arduino Mega2560"

Microcontroller = AVR Mega2560



Slut





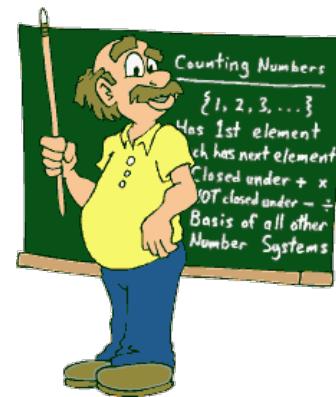
AARHUS
UNIVERSITY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

MSYS

Microcontroller Systems

Lektion 2_2: Talsystemer



Slides from: Henning Hargaard

Positionstal

Et par eksempler:

			417						30752
4	1	7		3	0	7	5	2	
400	10	7		30000	0	700	50	2	
4*100	1*10	7*1		3*10000	0*1000	7*100	5*10	2*1	
4*10 ²	1*10 ¹	7*10 ⁰		3*10 ⁴	0*10 ³	7*10 ²	5*10 ¹	2*10 ⁰	

Positionstal, 10-talsystemet

- Man kan opskrive en simpel formel for tallenes decimalværdi:

t: 30752

Lad $t.i$ betegne det i 'te ciffer i tallet t
og N antal cifre i tallet:

$t.4$	$t.3$	$t.2$	$t.1$	$t.0$
3	0	7	5	2

$3 \cdot 10^4 \quad 0 \cdot 10^3 \quad 7 \cdot 10^2 \quad 5 \cdot 10^1 \quad 2 \cdot 10^0$

decimal værdi = $(\sum i \mid 0 \leq i < N : t.i \cdot 10^i)$

10 symboler: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 (cifre)

Positionstal, 2-talsystemet

2 symboler: 0, 1 (bit: binary digit)

t: 10011

t.4	t.3	t.2	t.1	t.0
1	0	0	1	1

$1 * 2^4$	$0 * 2^3$	$0 * 2^2$	$1 * 2^1$	$1 * 2^0$
16	0	0	2	1

decimal værdi = $(\sum i \mid 0 \leq i < N : t.i * 2^i)$

$10011_2 = 19_{10}$

g-talsystemet

g symboler: 0, 1, ..., g-1 (cifre) kaldes også for: radix

$$\text{decimal værdi} = (\sum i \mid 0 \leq i < N : t.i \cdot g^i)$$

g = 8: 0, 1, 2, 3, 4, 5, 6, 7

g = 16: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

g = 16, kaldes også for: hexadecimal

$$\begin{aligned} ABE_{16} &= A \cdot 16^2 + B \cdot 16^1 + E \cdot 16^0 \\ &= 10 \cdot 16^2 + 11 \cdot 16^1 + 14 \cdot 16^0 \\ &= 2560 + 176 + 14 \\ &= 2750 \end{aligned}$$

Decimale og binære tal

Converting from binary to decimal

To convert from binary to decimal, it is important to understand the concept of weight associated with each digit position. First, as an analogy, recall the weight of numbers in the base 10 system, as shown in the diagram. By the same token, each digit position of a number in base 2 has a weight associated with it:

740683_{10}	=	
3×10^0	=	3
8×10^1	=	80
6×10^2	=	600
0×10^3	=	0000
4×10^4	=	40000
7×10^5	=	<u>700000</u>
		740683

110101_2	=	<i>Decimal</i>	<i>Binary</i>
1×2^0	=	1×1	= 1
0×2^1	=	0×2	= 0
1×2^2	=	1×4	= 4
0×2^3	=	0×8	= 0
1×2^4	=	1×16	= 16
1×2^5	=	1×32	= <u>32</u>
			110101
		53	

Test ("socrative.com"): Room = AUQUIZ

- Hvordan skrives det binære tal: 00010100 som decimalt tal ?

A: 40

B: 10

C: 20

D: 5

Fra decimal til binær

Example 0-1

Convert 25_{10} to binary.

Solution:

	Quotient	Remainder	
$25/2 =$	12	1	LSB (least significant bit)
$12/2 =$	6	0	
$6/2 =$	3	0	
$3/2 =$	1	1	
$1/2 =$	0	1	MSB (most significant bit)

Therefore, $25_{10} = 11001_2$.

Test ("socrative.com"): Room = AUQUIZ

- Hvordan skrives det decimale tal 217 som binært tal ?

A: 00101100

B: 01101100

C: 11000001

D: 11011001

Hexa-decimale tal

Binary	Decimal	Hexadecimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

Bemærk:
A = 10
B = 11
C = 12
D = 13
E = 14
F = 15

Eksempler med HEX-tal

Example 0-4

Represent binary 100111110101 in hex.

Solution:

First the number is grouped into sets of 4 bits: 1001 1111 0101.

Then each group of 4 bits is replaced with its hex equivalent:

1001	1111	0101
9	F	5

Therefore, $100111110101_2 = 9F5$ hexadecimal.

Example 0-5

Convert hex 29B to binary.

Solution:

2	9	B
29B	=	0010 1001 1011

Dropping the leading zeros gives 1010011011.



Test ("socrative.com"): Room = AUQUIZ

- Hvordan skrives det decimale tal 217 som "hex" tal ?
A: 17
B: 6C
C: D9
D: C1

Binær addition

Table 0-3: Binary Addition

A + B	Carry	Sum
0 + 0	0	0
0 + 1	0	1
1 + 0	0	1
1 + 1	1	0

Example 0-8

Add the following binary numbers. Check against their decimal equivalents.

Solution:

<i>Binary</i>	<i>Decimal</i>
1101	13
+	
1001	9
10110	22

Binær addition

$$\begin{array}{r} X \\ Y \\ \hline X + Y \end{array} \quad \begin{array}{r} 173 \\ + 44 \\ \hline 217 \end{array}$$

A binary addition diagram showing the sum of 173 and 44. The result 217 is circled in red. Blue arrows point from the carry bits of the sum to the carry bit of the next column.

1	0	1	1	0	1	1	1	0	1
+	0	0	1	0	1	1	1	1	0

1	1	0	1	1	1	1	0	0	1

$$\begin{array}{r} X \\ Y \\ \hline X + Y \end{array} \quad \begin{array}{r} 190 \\ + 141 \\ \hline 331 \end{array}$$

A binary addition diagram showing the sum of 190 and 141. The result 331 is circled in red. Blue arrows point from the carry bits of the sum to the carry bit of the next column.

1	0	1	1	1	1	1	0		
+	1	0	0	0	1	1	0	1	1

1	0	1	0	0	1	0	1	1	1

Angivelse af valgt talsystem i Microchip Studio

- **0b** betyder binært (2-tal-systemet)
Eksempel: **0b**01101011
- **0x** betyder hexadecimalt (16-tal-systemet)
Eksempel: **0x**6B
- ”Ingenting” betyder decimalt (10-tal-systemet)
Eksempel: 107

Bemærk: 0b01101011 = 0x6B = 107

Det er blot vores måde at skrive tallet på, der er forskelligt i eksemplerne.

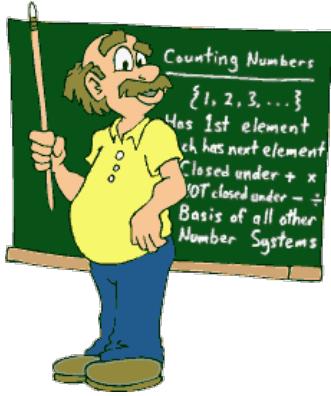
Slut





AARHUS
UNIVERSITY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING



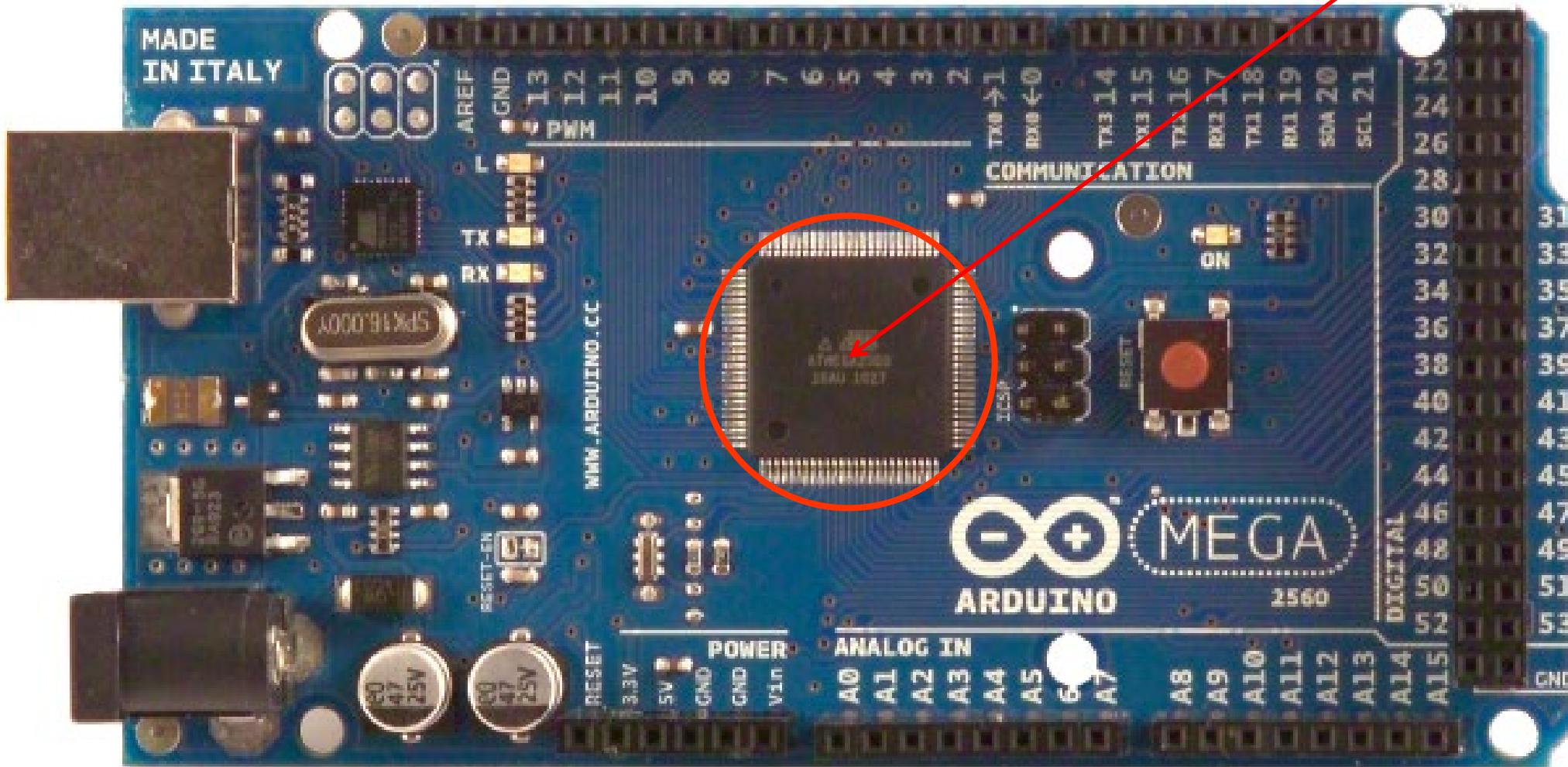
MSYS

Microcontroller Systems

Lektion 3_1: Arduino Mega2560

"Arduino Mega2560"

Microcontroller = ATmega2560



Husk at boardet ikke må opfattes som "vores computer" !

Den monterede microcontroller er "vores computer" !

TX/RX
Leds

"Test"
Led 13

digital pins

I2C

Power
Led

USB Interface

External
Power
Supply

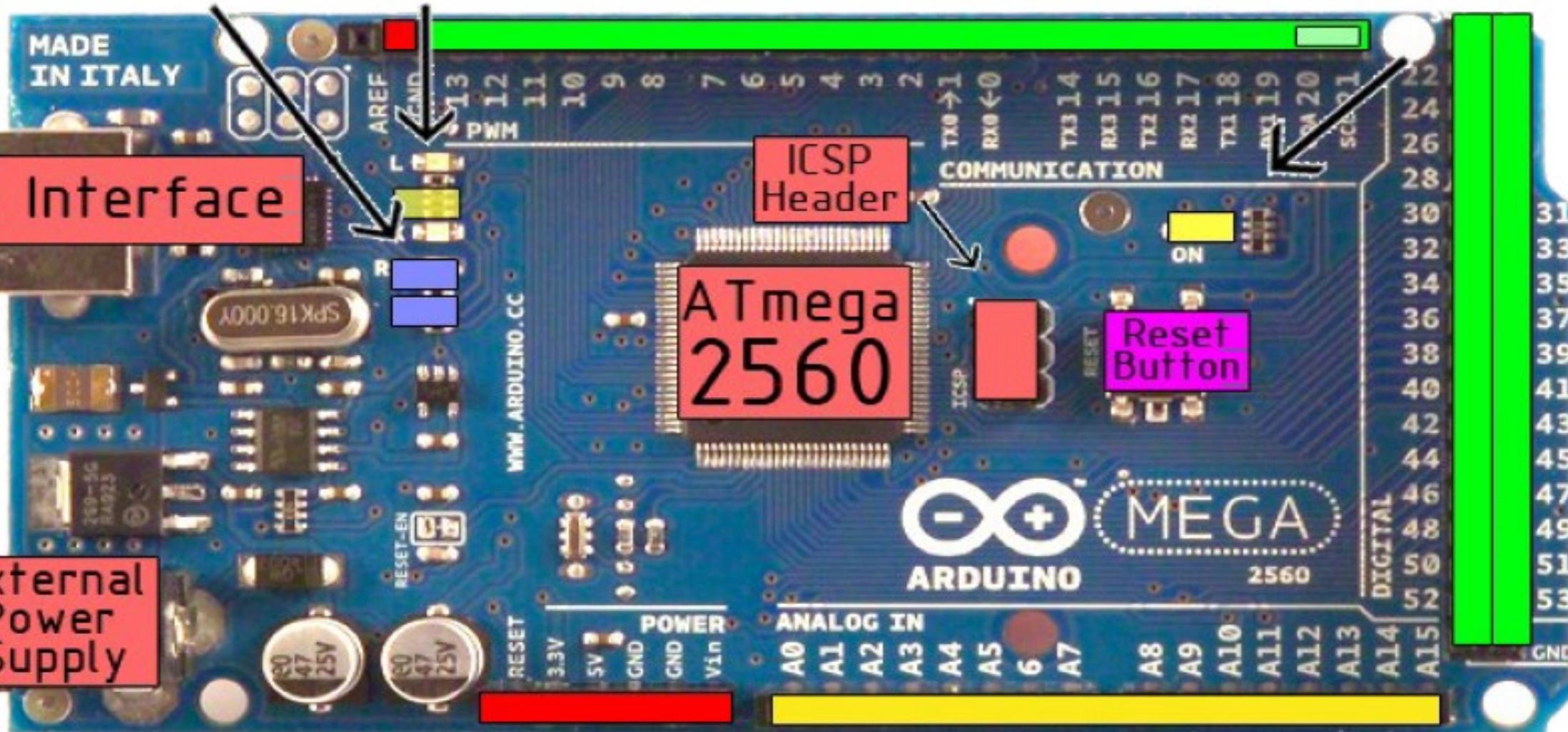
ICSP
Header

ATmega
2560

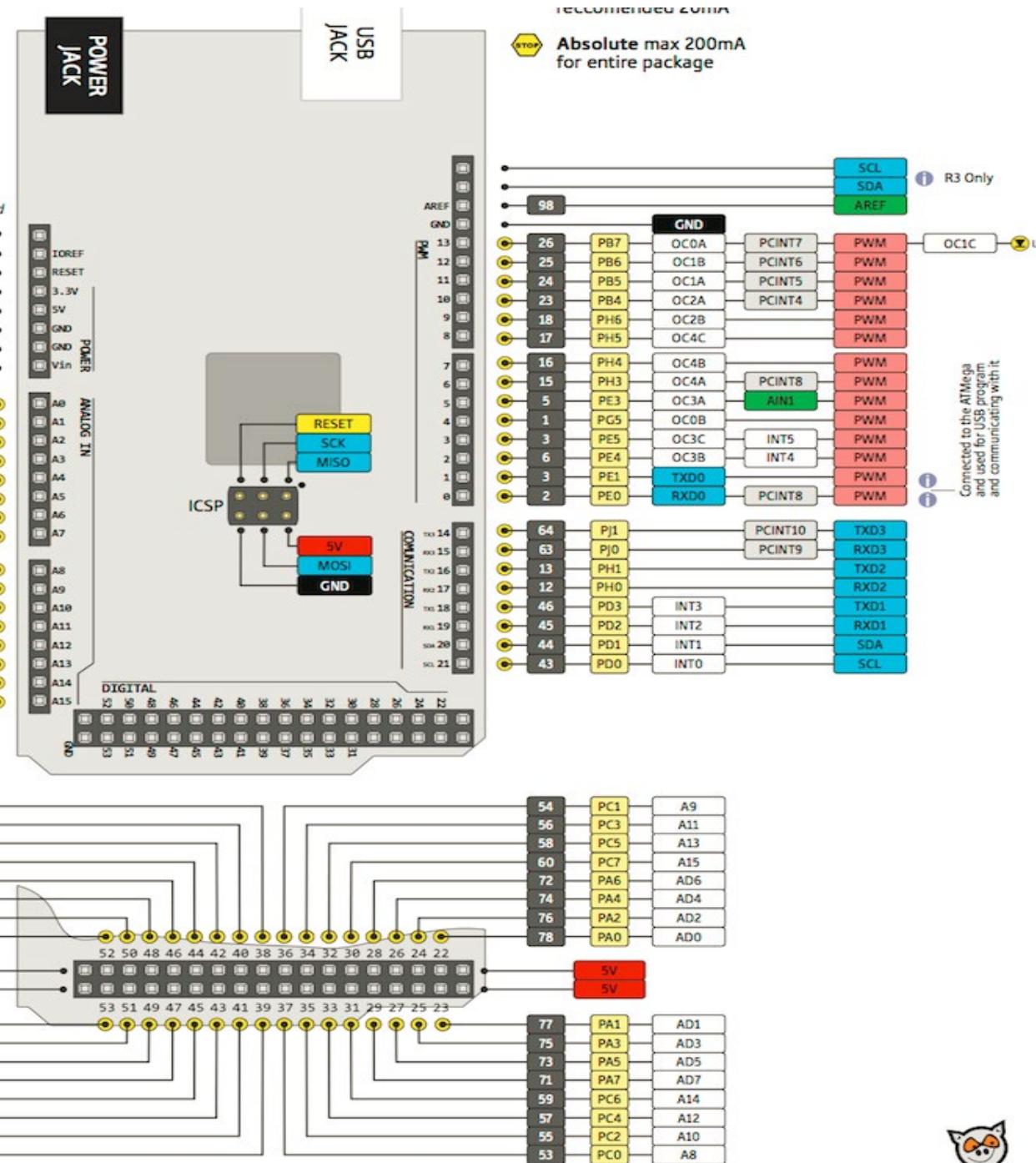
Reset
Button

power pins

analog pins



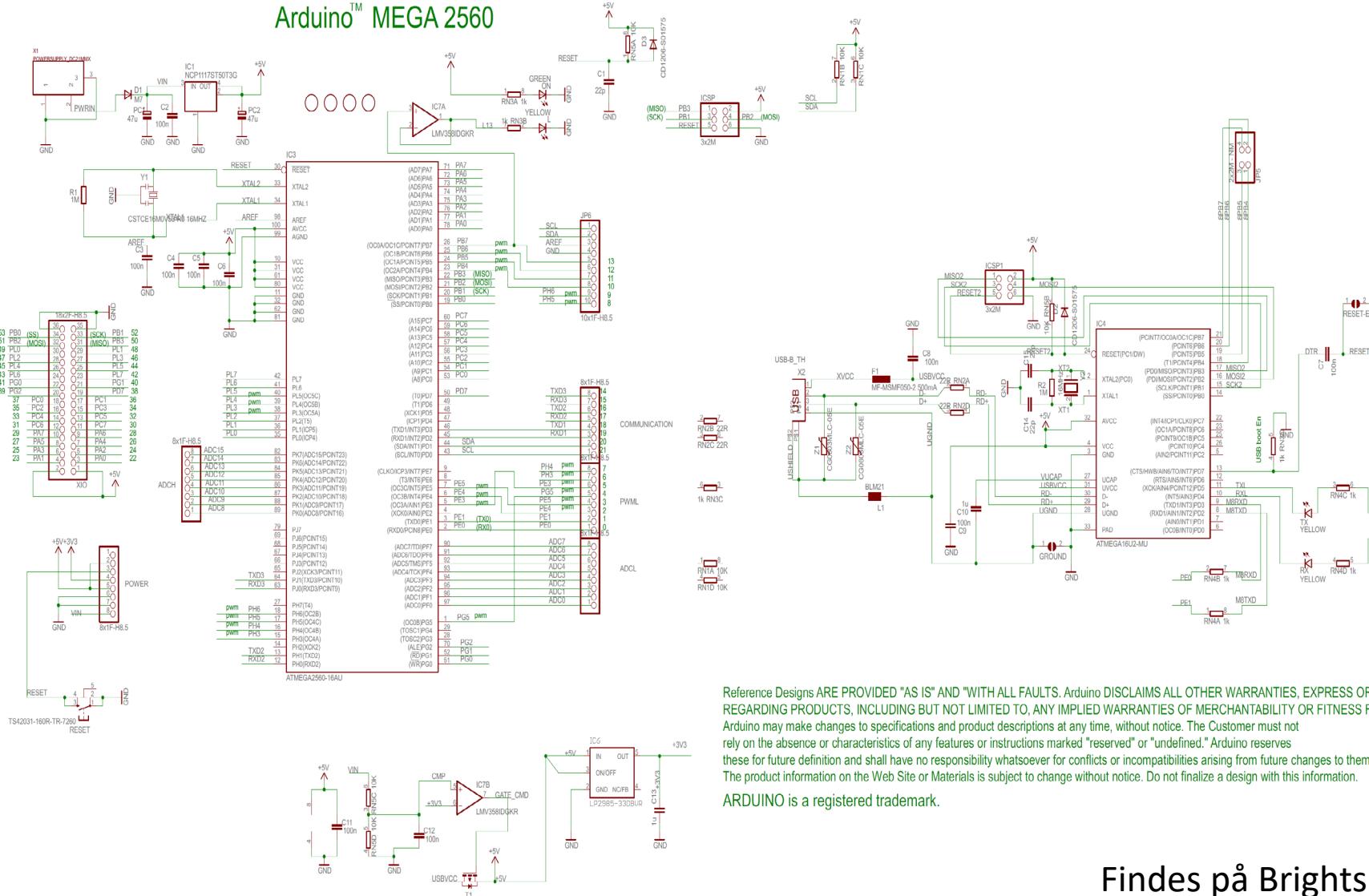
Pinouts er på MSYS Brightspace



"Arduino Mega2560"

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	54 (of which 14 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz

Diagram for "Arduino Mega2560"



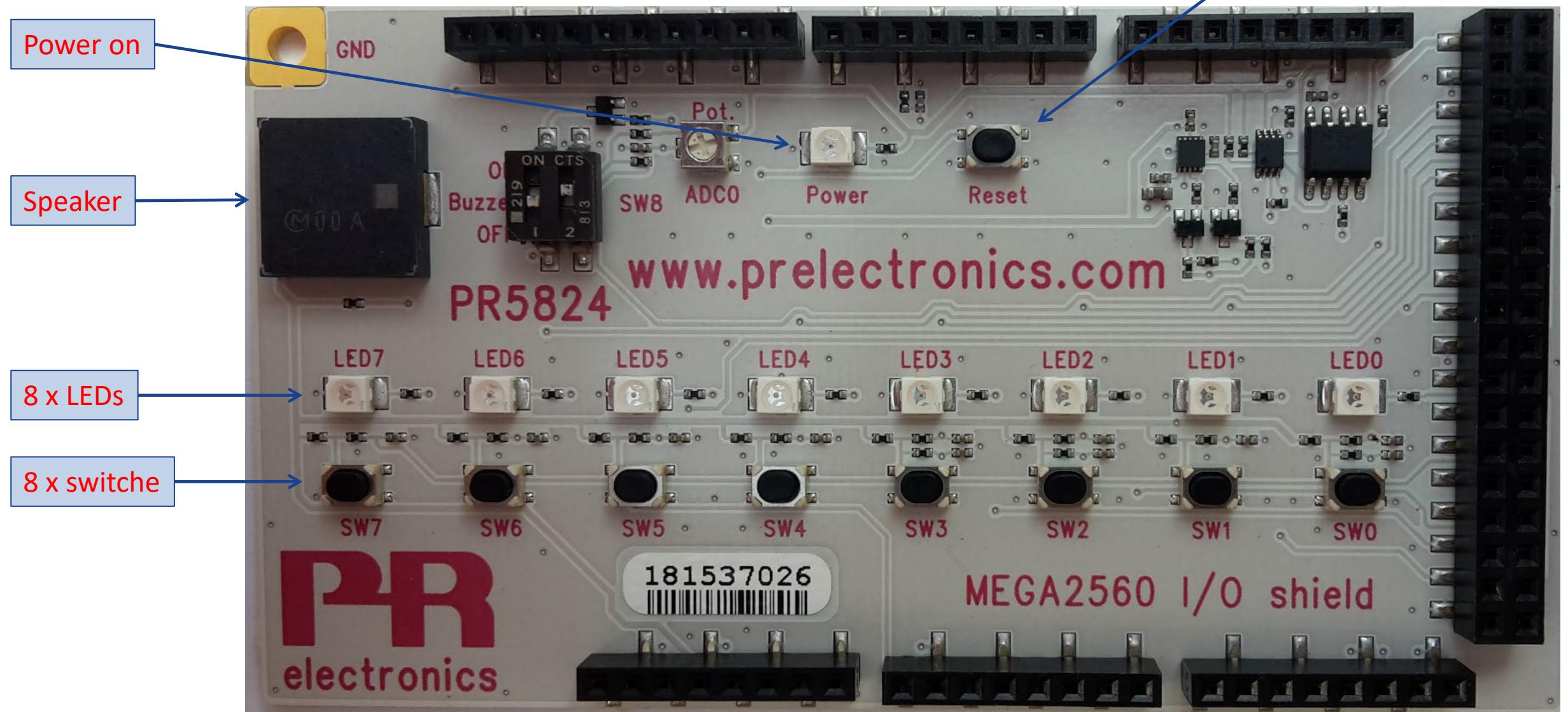
Reference Designs ARE PROVIDED "AS IS" AND "WITH ALL FAULTS. Arduino DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, REGARDING PRODUCTS, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Arduino may make changes to specifications and product descriptions at any time, without notice. The Customer must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Arduino reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

The product information on the Web Site or Materials is subject to change without notice. Do not finalize a design with this information.

ARDUINO is a registered trademark.

Findes på Brightspace!

Mega2560 I/O Shield (version 3)



"Mega2560 I/O Shield"

- Power on LED (rød)
- Reset knap
- 8 lysdioder, LED..LED0 (PORTB)
- 8 trykknapper, SW7..SW0 (PORTA)
- Lydgiver/speaker (buzzer) (PORTB, bit 5)
- On/Off jumper eller DIP switch for lydgiver
- Potentiometer for A/D konverter (ADC0 = PF0)
- Low pass filter for PWM -> Analog
- 2 interrupt trykknapper (INT3 = SW3, INT4 = SW4)
- 1 trykknap for counting (T0 = PD7 = SW0)
- 1 digital temperaturmåler (LM75)
- 1 digital accelerometer (ICMA8652)

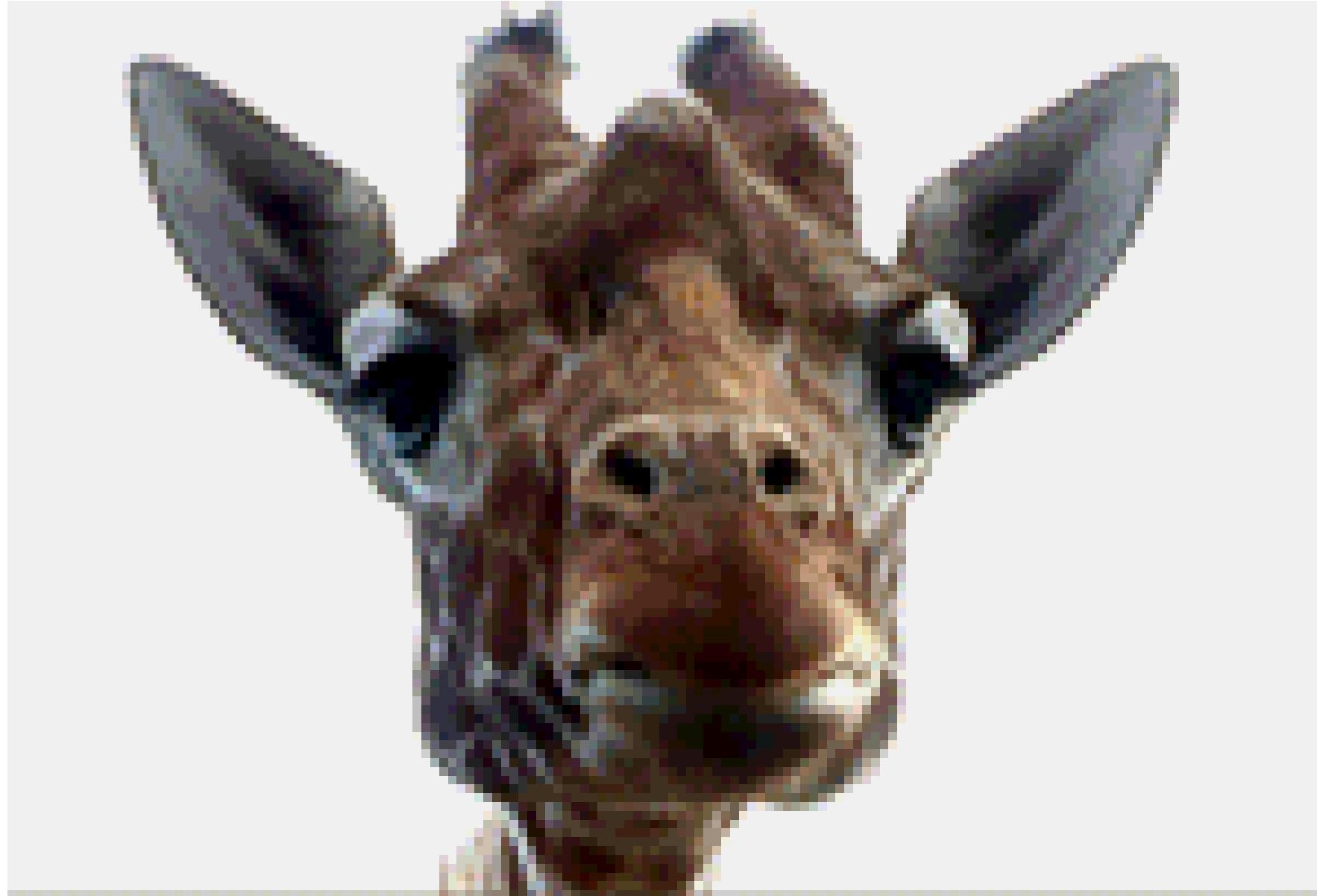
Kursus-materiale

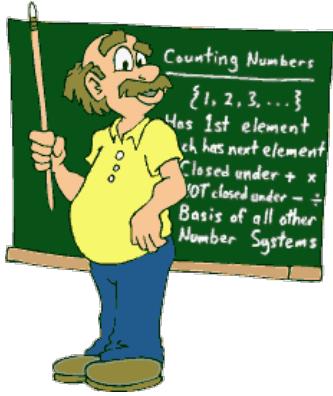
Mega32 / Mega2562

Arduino Mega2560 Board

Mega2560 I/O shield

Slut





MSYS

Microcontroller Systems

Lektion 3_2: Microchip Studio

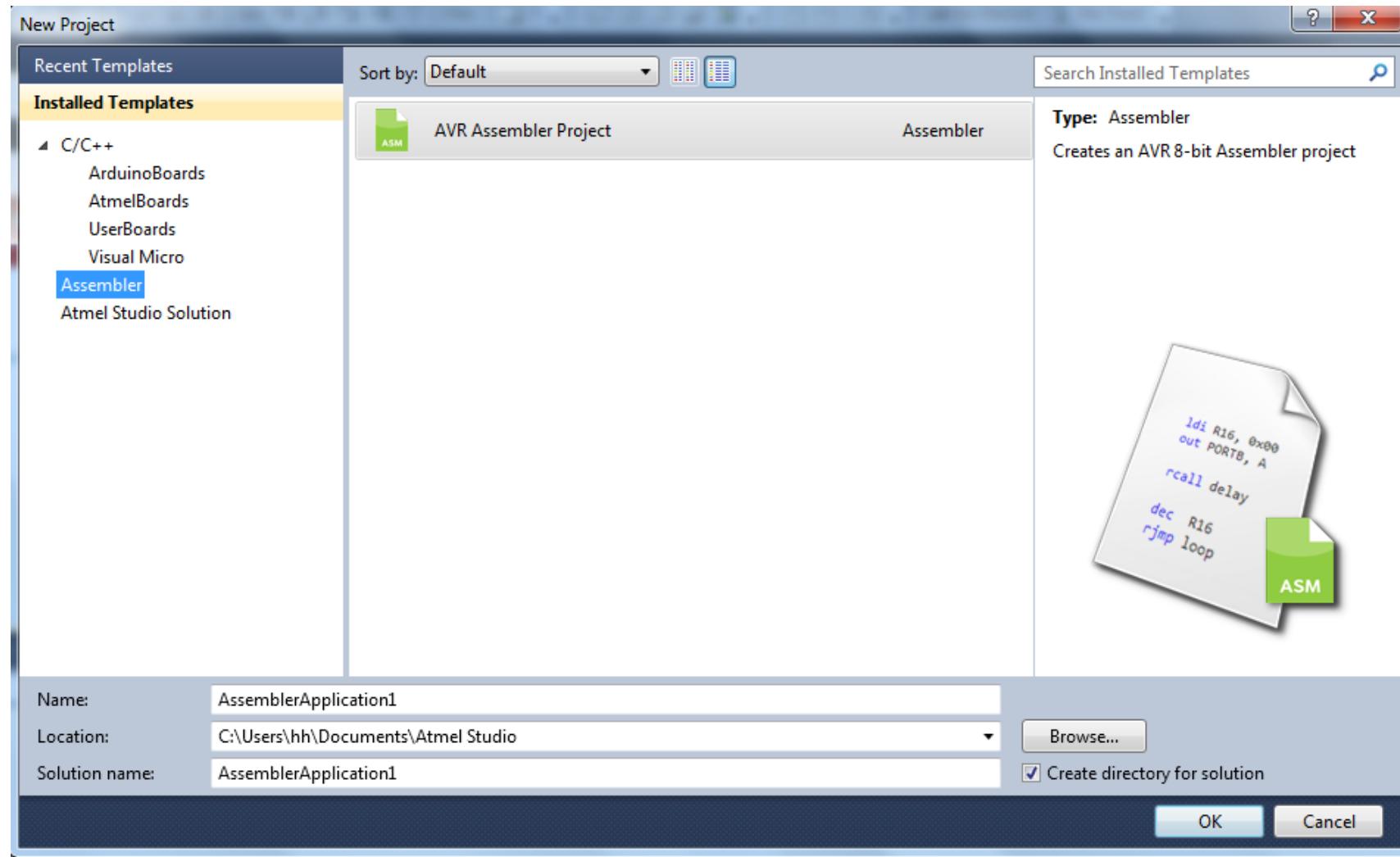
Microchip Studio 7



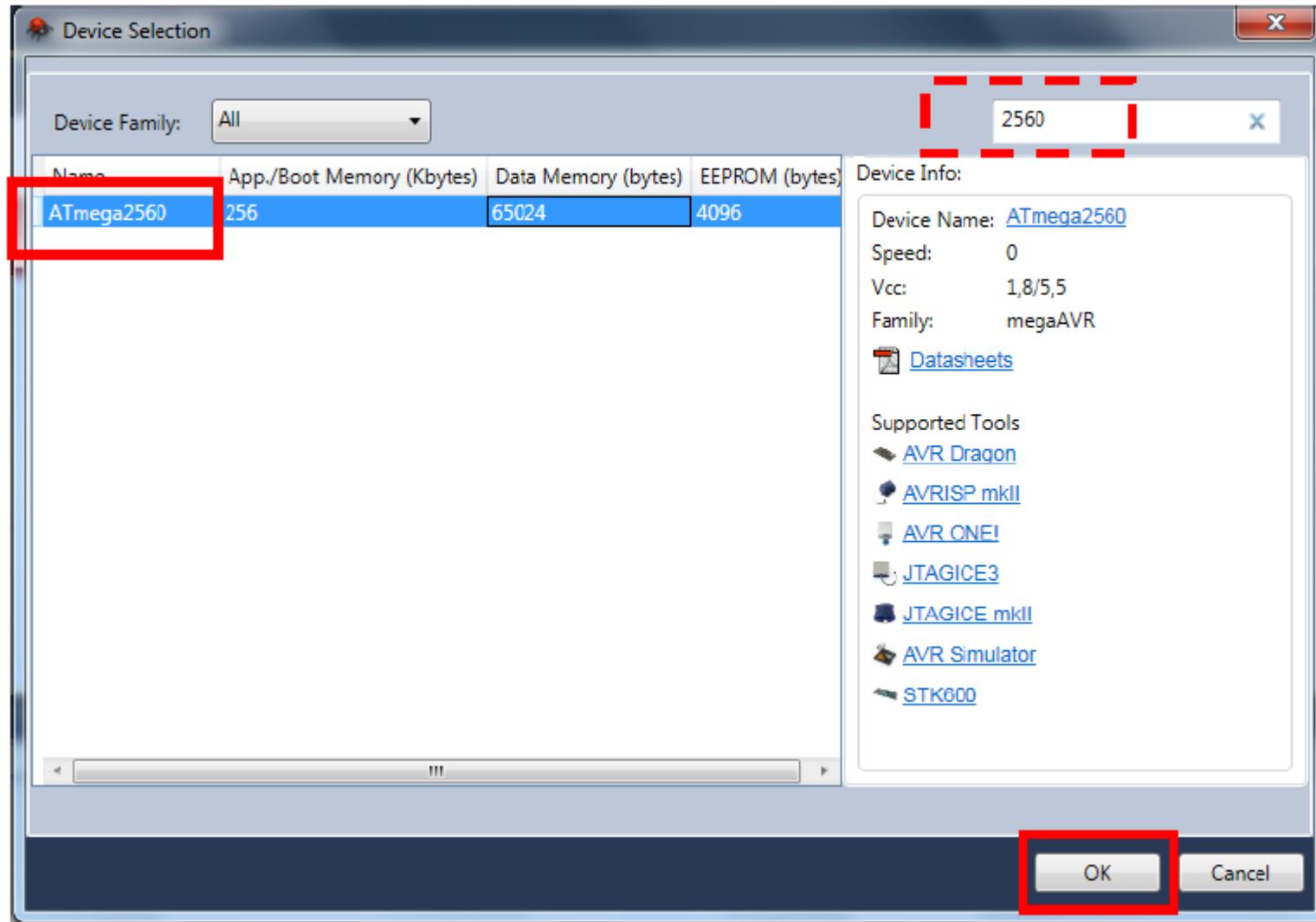
Glimrende (gratis) værktøj, som vi bruger til :

- Skrivning af assembly-programmer.
- Skrivning af C-programmer (eller C++).
- Download af program til "Arduino Mega2560".
- Test (simulering) af programmer.

Oprettelse af assembly projekt



Valg af microcontroller (ATmega2560)



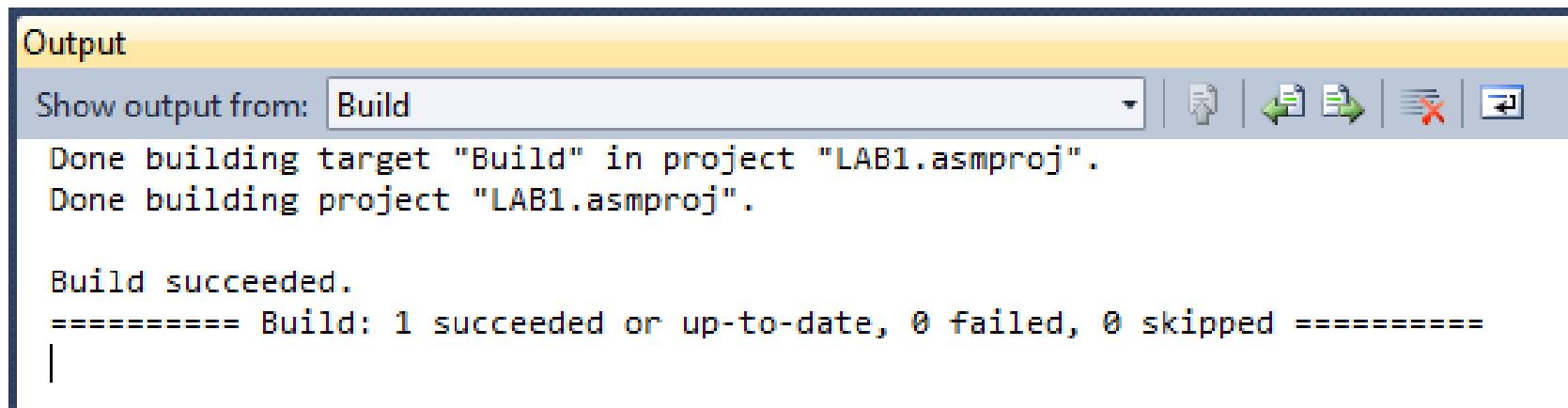
Skriv assembly koden

LAB1.asm X

```
;***** MSYS, LAB1 *****
;***** Henning Hargaard *****
;***** 14.august 2015 *****
;***** INITIERING *****
LDI R16,HIGH(RAMEND) ;Initialize Stack Pointer
OUT SPH,R16
LDI R16,LOW(RAMEND)
OUT SPL,R16
SER R16 ;PORTB = Outputs
OUT DDRB,R16

;***** PROGRAM-LOOP *****
CLR R16
LOOP:
LDI R17,9 ;R17 = 9
ADD R16,R17 ;R16 = R16 + R17
CALL DISP_AND_DELAY ;Display R16
JMP LOOP ;Jump to "LOOP"
```

Build (F7)

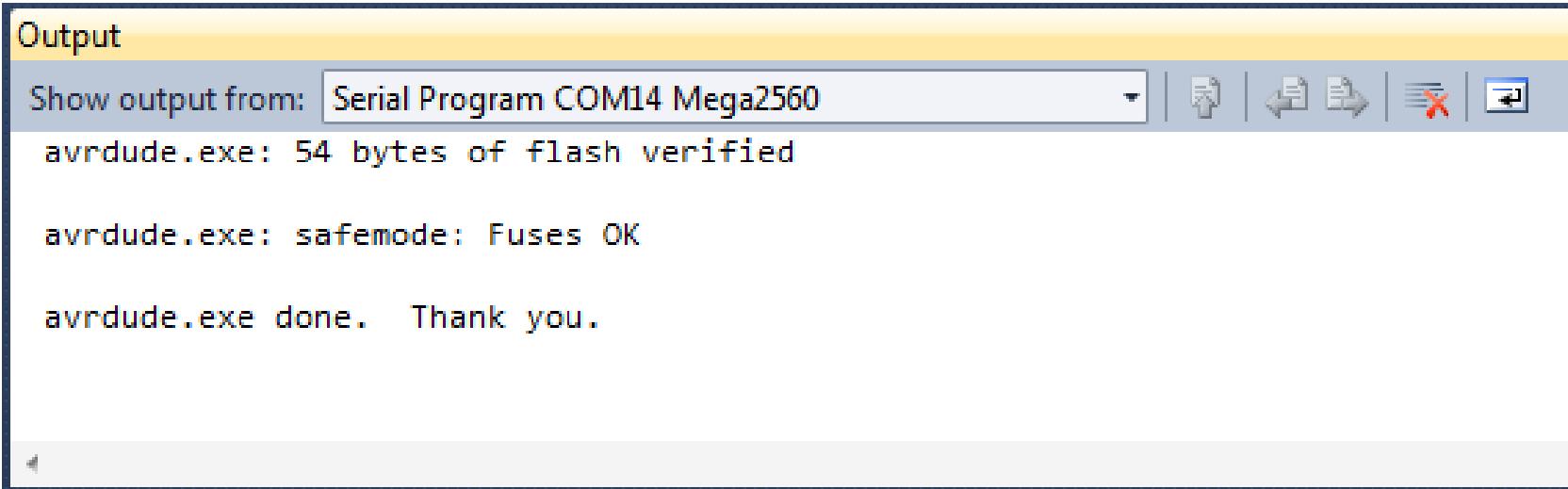


The screenshot shows the MSYS terminal window with the title "Output". The "Show output from:" dropdown is set to "Build". The terminal displays the following text:
Done building target "Build" in project "LAB1.asmproj".
Done building project "LAB1.asmproj".

Build succeeded.
===== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped =====

Herved assembleres koden (og vi
får dannet en "hex-fil" med
maskinkoderne)

Programmering af Mega2560



The screenshot shows the AVRDUDE output window with the following text:
Show output from: Serial Program COM14 Mega2560
avrduude.exe: 54 bytes of flash verified
avrduude.exe: safemode: Fuses OK
avrduude.exe done. Thank you.

Simulering (debug)

```
LAB1.asm X

;***** MSYS, LAB1 *****
;***** Henning Hargaard *****
;***** 14.august 2015 *****
;***** *****
;  
  
;***** INITIERING *****
LDI R16,HIGH(RAMEND) ;Initialize Stack Pointer
    OUT SPH,R16
    LDI R16,LOW(RAMEND)
    OUT SPL,R16
    SER R16           ;PORTB = Outputs
    OUT DDRB,R16  
  
;***** PROGRAM-LOOP *****
    CLR R16
LOOP:
    LDI R17,9          ;R17 = 9
    ADD R16,R17         ;R16 = R16 + R17
    CALL DISP_AND_DELAY ;Display R16
    JMP LOOP            ;Jump to "LOOP"
```

Debug control

Windows	1	▶
▶ Continue	2 F5	
■ Break All	3 Ctrl+Alt+Break	
■ Stop Debugging	4 Shift+F5	
■ Detach All	5	
Terminate All	6	
■ Restart	7 Ctrl+Shift+F5	
■ Reset	8	
■ Attach to Process...	9	
Exceptions...	10 Ctrl+Alt+E	
■ Step Into	11 F11	
■ Step Over	12 F10	
■ Step Out	13 Shift+F11	
■ QuickWatch...	14 Shift+F9	
Toggle Breakpoint	15 F9	
New Breakpoint	16	▶

Registre (R0 – R31)

Name	Value
Program Counter	0x00000001
Stack Pointer	0x21FF
X Register	0x0000
Y Register	0x0000
Z Register	0x0000
Status Register	1 T H S V N Z C
Cycle Counter	1
Frequency	16,000 MHz
Stop Watch	0,06 µs
Registers	
R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0x21
R17	0x00

Watch vindue

Name	Value	Type
PORTB	24	dword
R17	242	byte{reg}
R18	0	byte{reg}
R19	10	byte{reg}

I/O View

The screenshot shows the I/O View window of the MSYS - Microcontroller Systems software. The window title is "I/O View". The menu bar includes "LAB1.asm", "IO View", "ASF Explorer", "Solution Explorer", and "Properties". The "IO View" tab is selected.

The left pane displays a tree view of I/O components:

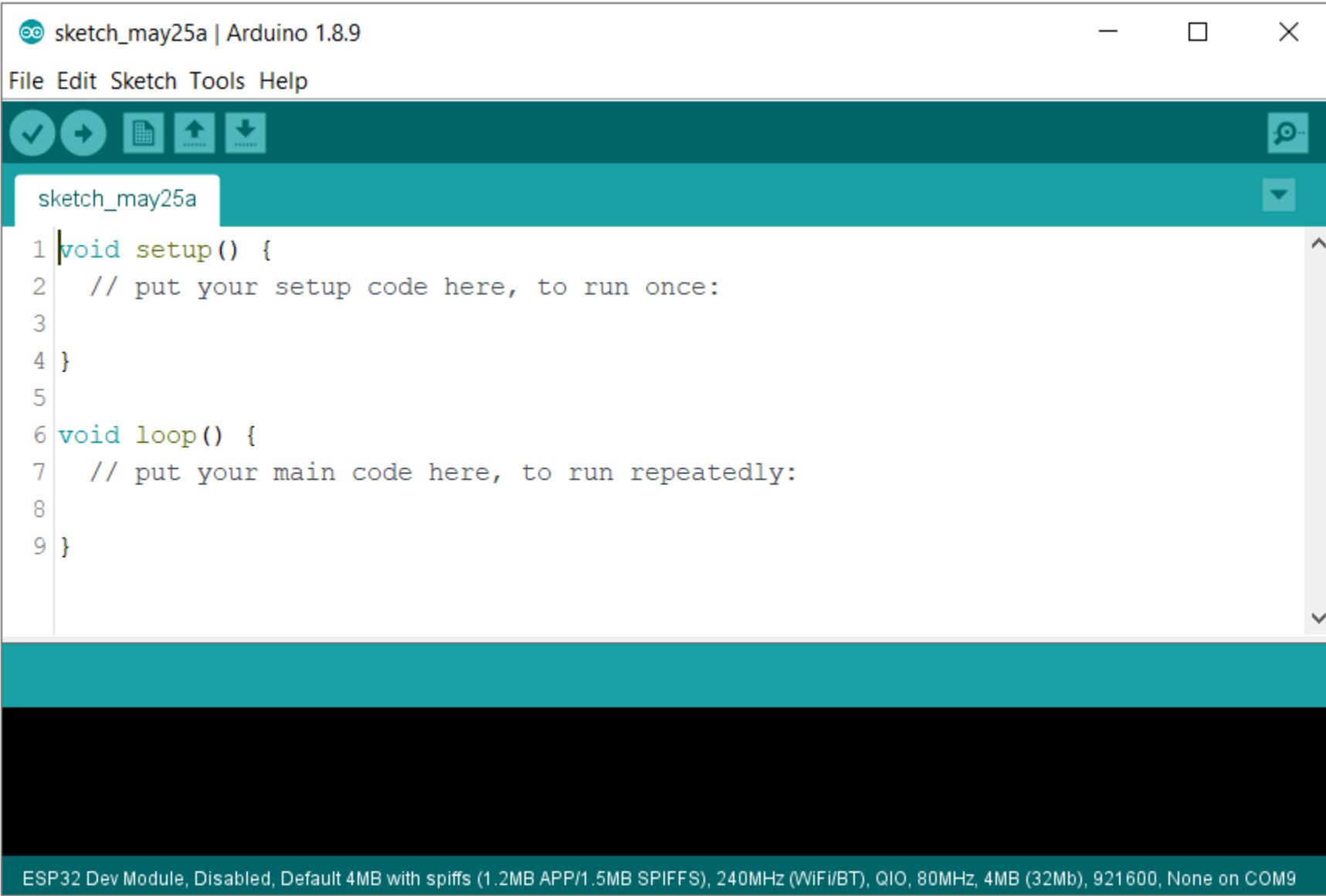
- CPU
- EEPROM
- EXTERNAL_INTERRUPT
- JTAG
- I/O PORTA
- I/O PORTB
- I/O PORTC
- I/O PORTD
- I/O PORTE
- I/O PORTF
- I/O PORTG

The "PORTB" node is currently selected, highlighted with a blue background.

The right pane contains two tables:

Name	Address	Value	Bits
I/O PINB	0x23	0xF2	<input type="checkbox"/>
I/O DDRB	0x24	0xFF	<input type="checkbox"/>
I/O PORTB	0x25	0xF2	<input type="checkbox"/>

Arduino IDE = "No go" 😞



The screenshot shows the Arduino IDE interface with the following details:

- Title Bar:** sketch_may25a | Arduino 1.8.9
- Menu Bar:** File Edit Sketch Tools Help
- Toolbar:** Includes icons for save, upload, and refresh.
- Sketch Editor:** Displays the code for 'sketch_may25a'.

```
1 void setup() {
2     // put your setup code here, to run once:
3
4 }
5
6 void loop() {
7     // put your main code here, to run repeatedly:
8
9 }
```
- Status Bar:** Shows connection information: ESP32 Dev Module, Disabled, Default 4MB with spiffs (1.2MB APP/1.5MB SPIFFS), 240MHz (WiFi/BT), QIO, 80MHz, 4MB (32Mb), 921600, None on COM9

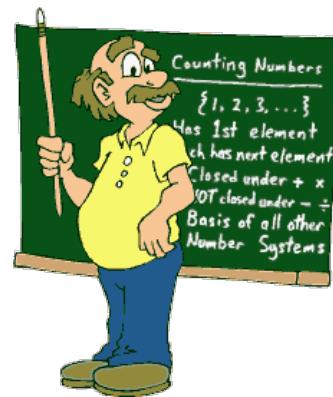
Slut



MSYS

Microcontroller Systems

Lektion 4: AVR arkitektur

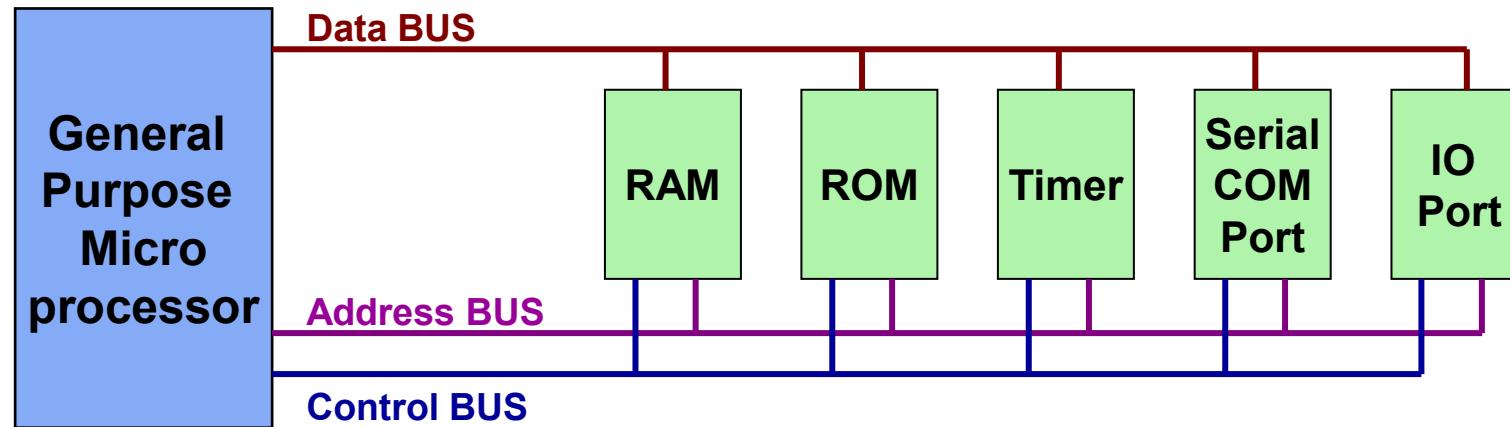


Slides from Henning Hargaard

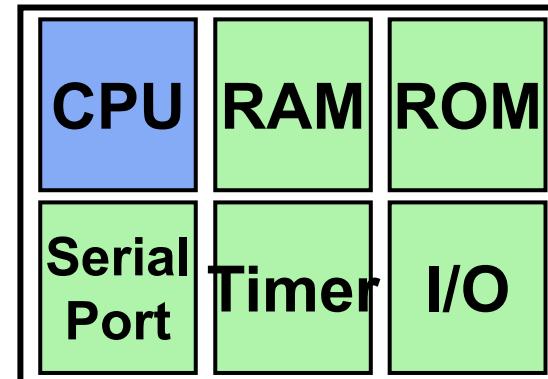
Michael Alrøe

General Purpose Microprocessors vs. Microcontrollers

- General Purpose Microprocessors



- Microcontrollers

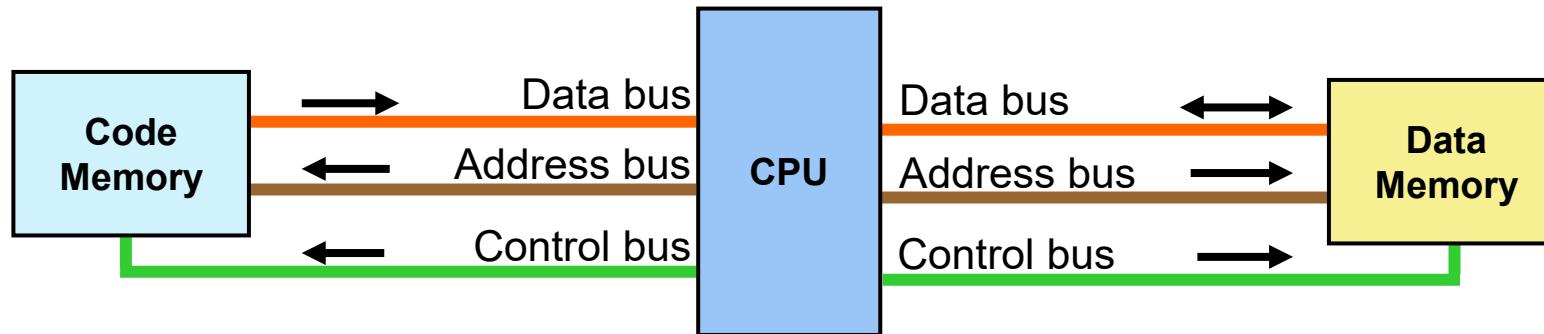


Common microcontrollers

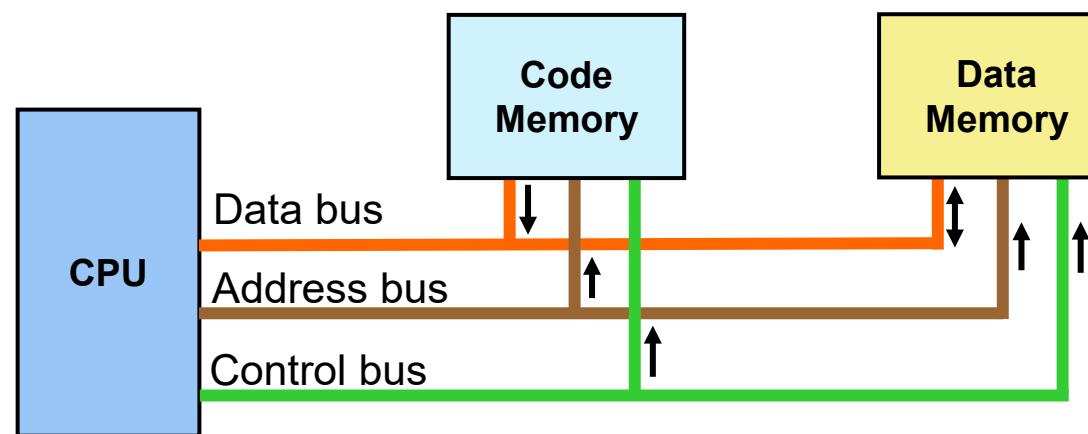
- 8-bit microcontrollers
 - AVR
 - PIC
 - HCS12
 - 8051
 -
- 32-bit microcontrollers
 - ARM
 - PIC32
 -

Von Neumann vs. Harvard arkitektur

- Harvard arkitektur:

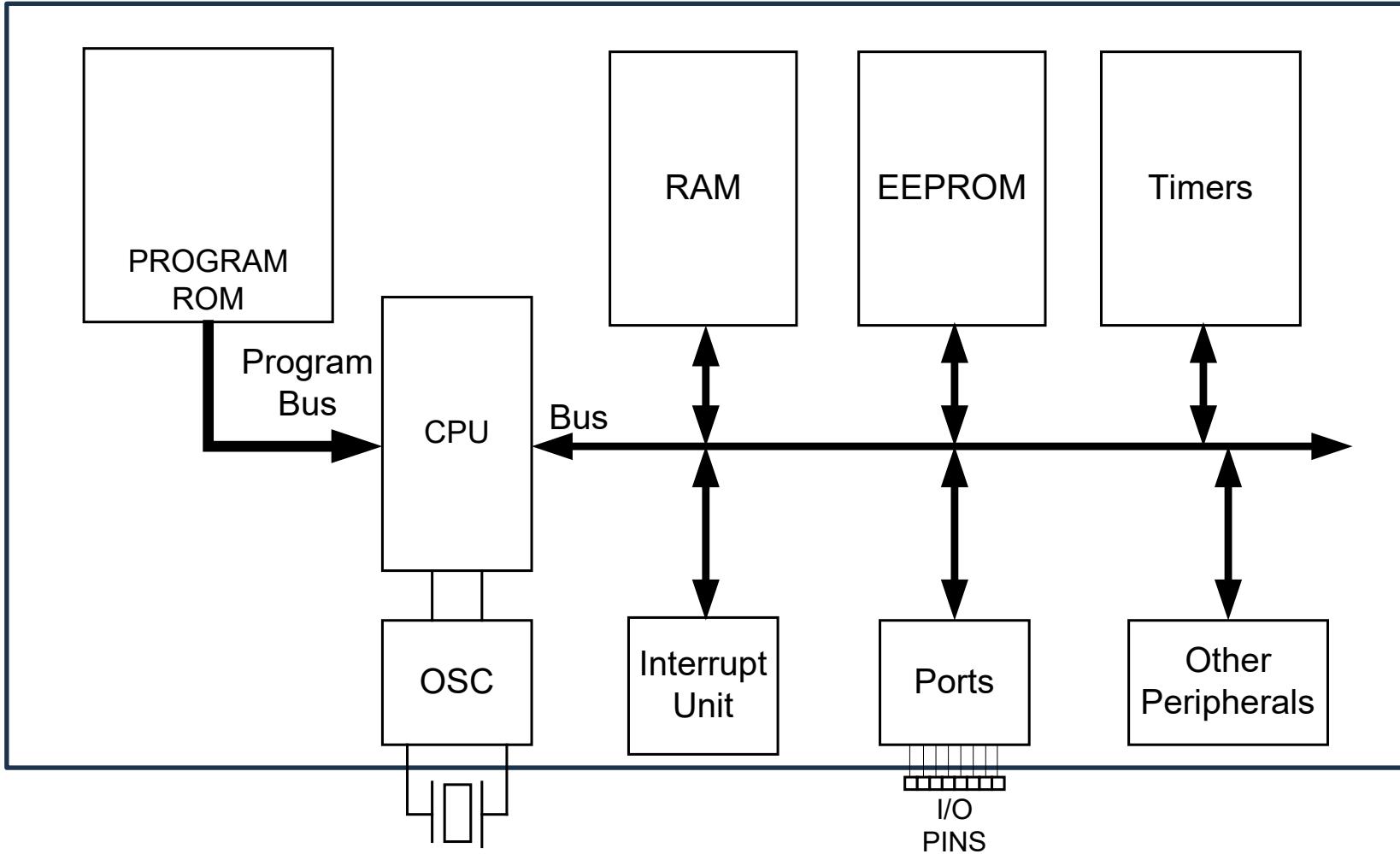


- Von Neumann arkitektur:



AVR internal architecture

AVR:



AVR different groups

- Classic AVR
 - e.g. AT90S2313, AT90S4433
- Mega
 - e.g. ATmega8, ATmega32, ATmega128

- Tiny

- e.g.

- Special

- e.g.

Table 1-4: Some Members of the Mega Family

Part Num	Code ROM	Data RAM	Data EEPROM	I/O pins pins	ADC	Timers	Pin numbers & Package
ATmega8	8K	1K	0.5K	23	8	3	TQFP32,PDIP28
ATmega16	16K	1K	0.5K	32	8	3	TQFP44,PDIP40
ATmega32	32K	2K	1K	32	8	3	TQFP44,PDIP40
ATmega64	64K	4K	2K	54	8	4	TQFP64,MLF64
ATmega1280	128K	8K	4K	86	16	6	TQFP100,CBGA

Notes:

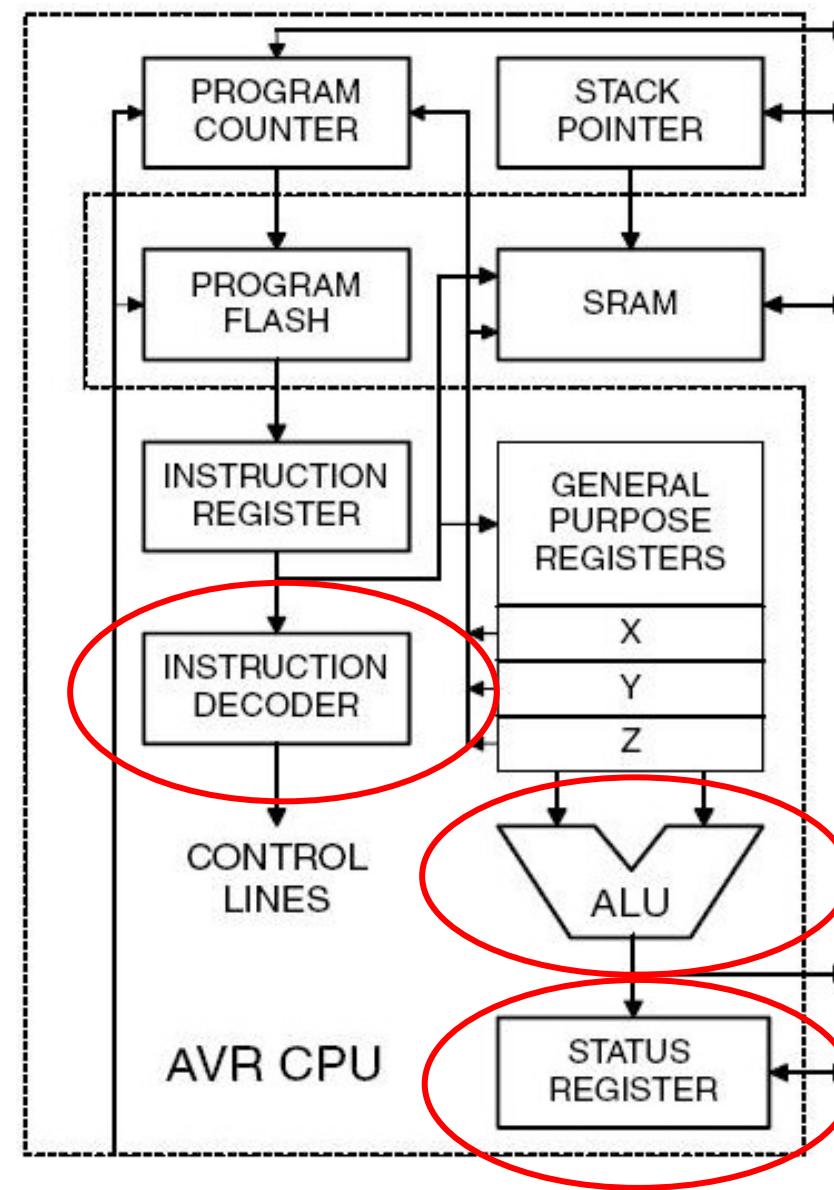
1. All ROM, RAM, and EEPROM memories are in bytes.
2. Data RAM (General-Purpose RAM) is the amount of RAM available for data manipulation (scratch pad) in addition to the Registers space.
3. All the above chips have USART for serial data transfer.

AVR: CPU

Instruktioner er
koder bestående af
0'er og 1'taller !

ALU foretager
beregninger.

Status register
ændres ved nogle
beregninger.

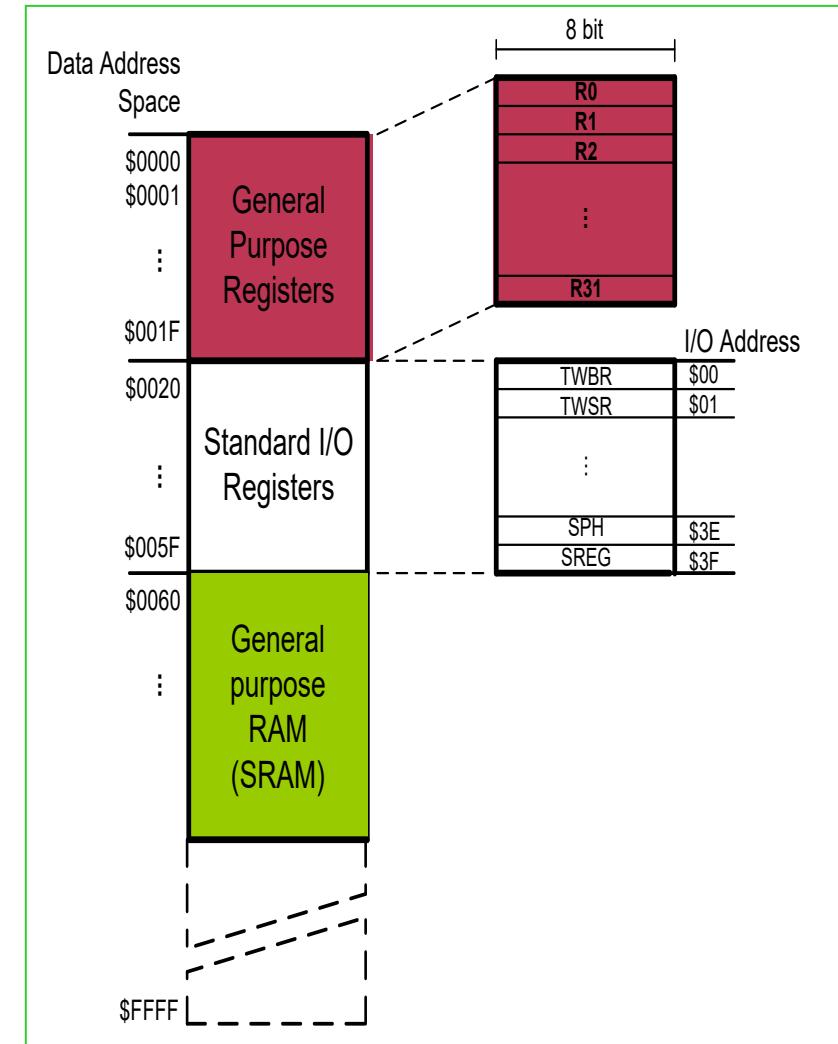
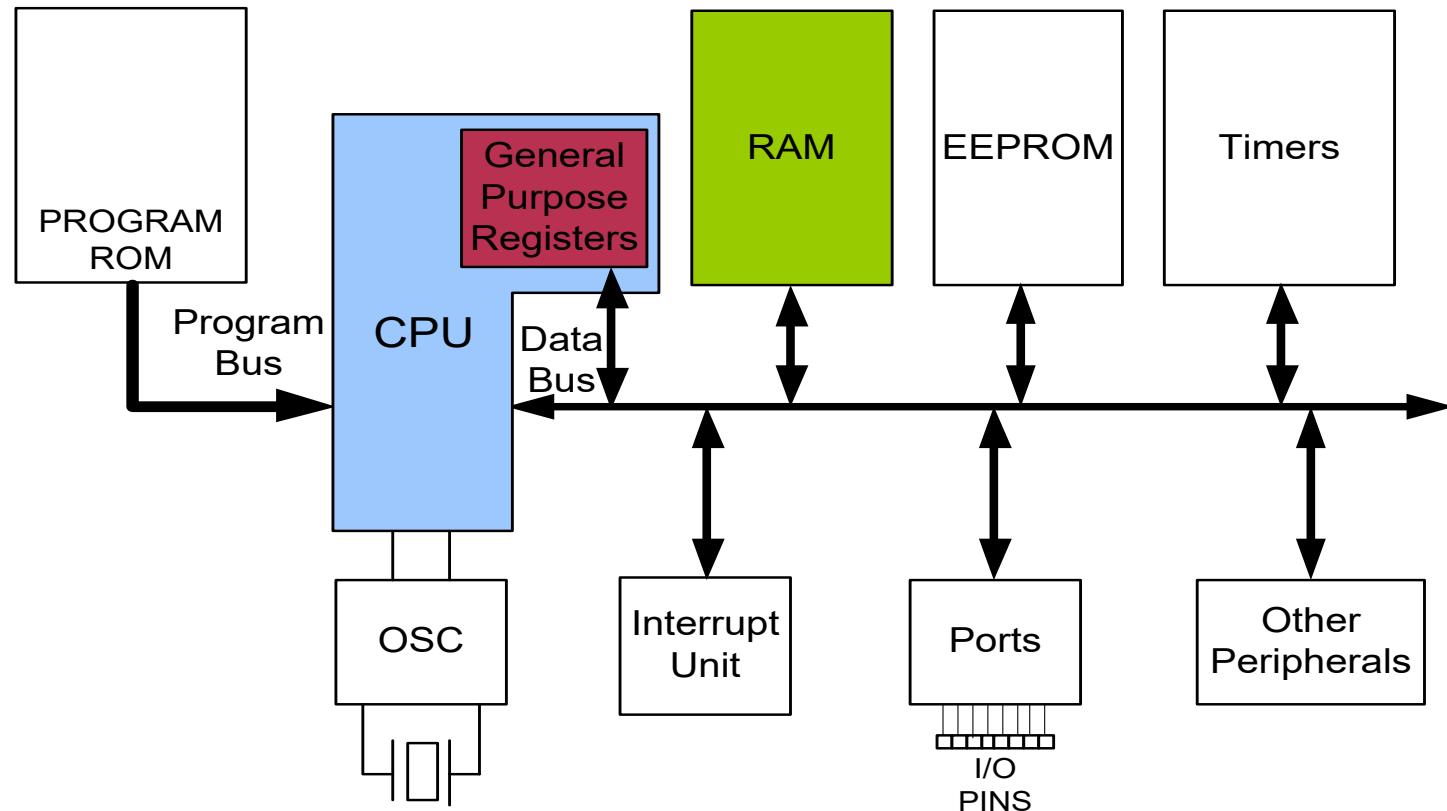


AVR: Arbejdsregistre

General Purpose Working Registers	7	0	Addr.	
			\$00	R0
			\$01	R1
			\$02	R2
				...
			\$0D	R13
			\$0E	R14
			\$0F	R15
			\$10	R16
			\$11	R17
				...
			\$1A	X-register Low Byte
			\$1B	X-register High Byte
			\$1C	Y-register Low Byte
			\$1D	Y-register High Byte
			\$1E	Z-register Low Byte
			\$1F	Z-register High Byte

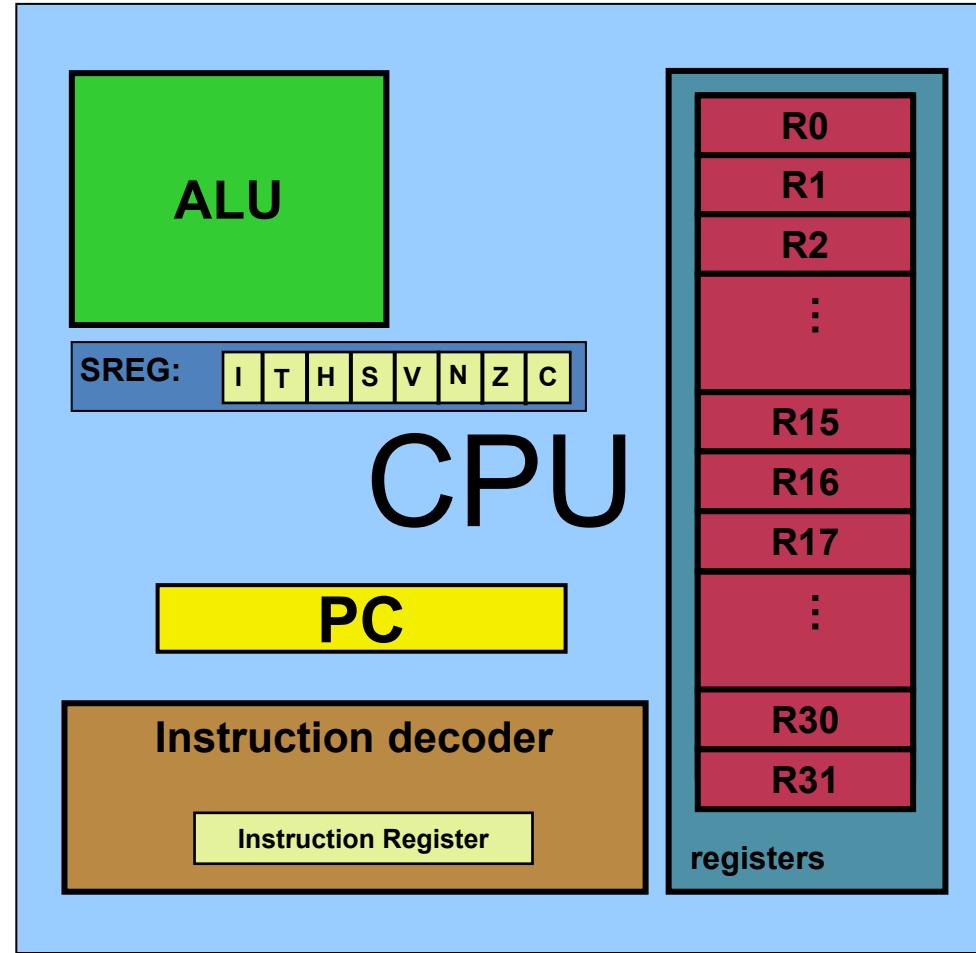
Kaldes også:
"General Purpose
Registre"

AVR: Data Address Space

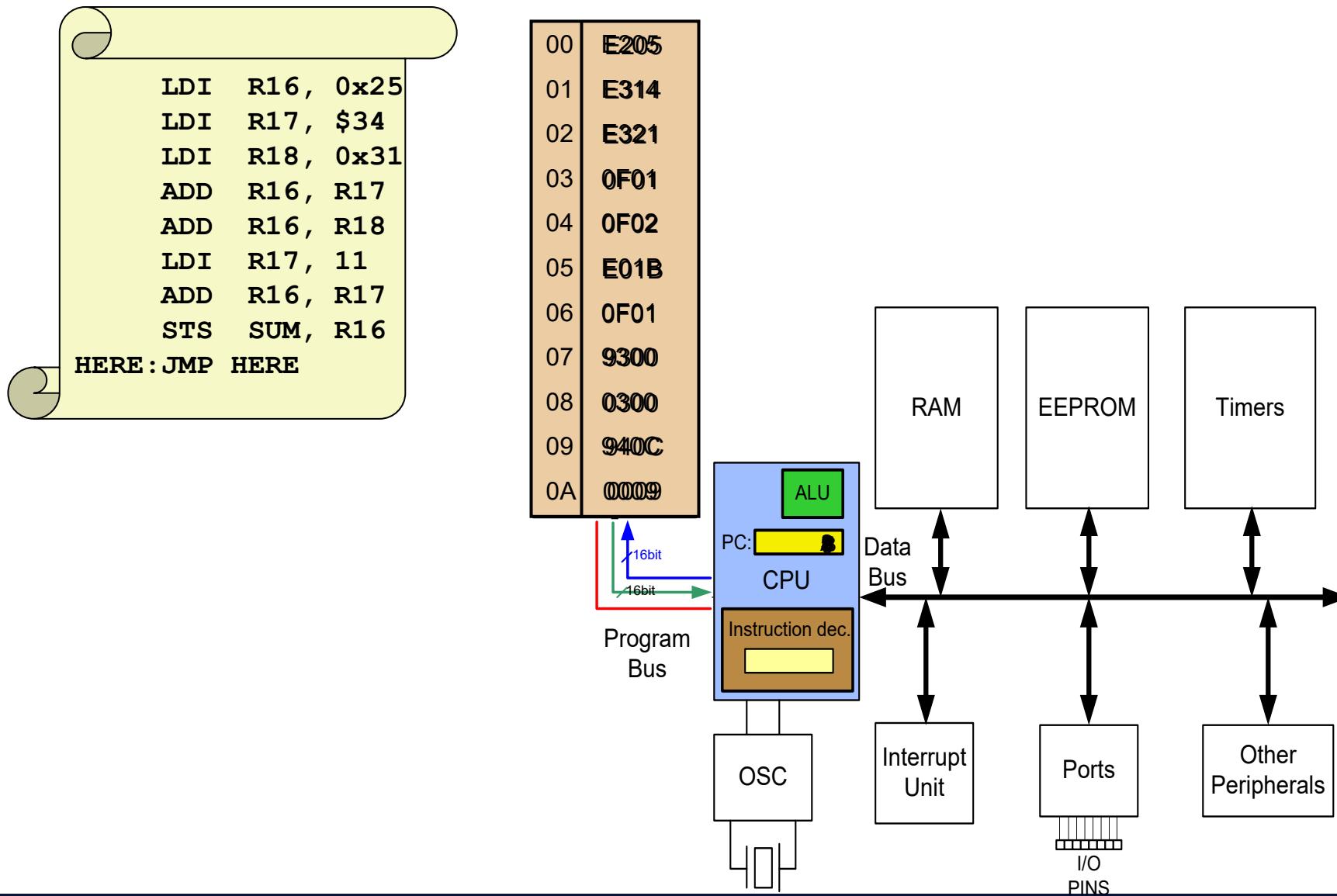


AVR: CPU (alternativ version)

- ALU
- 32 General Purpose registers (R0 to R31)
- Program Counter (PC) register
- Instruction decoder

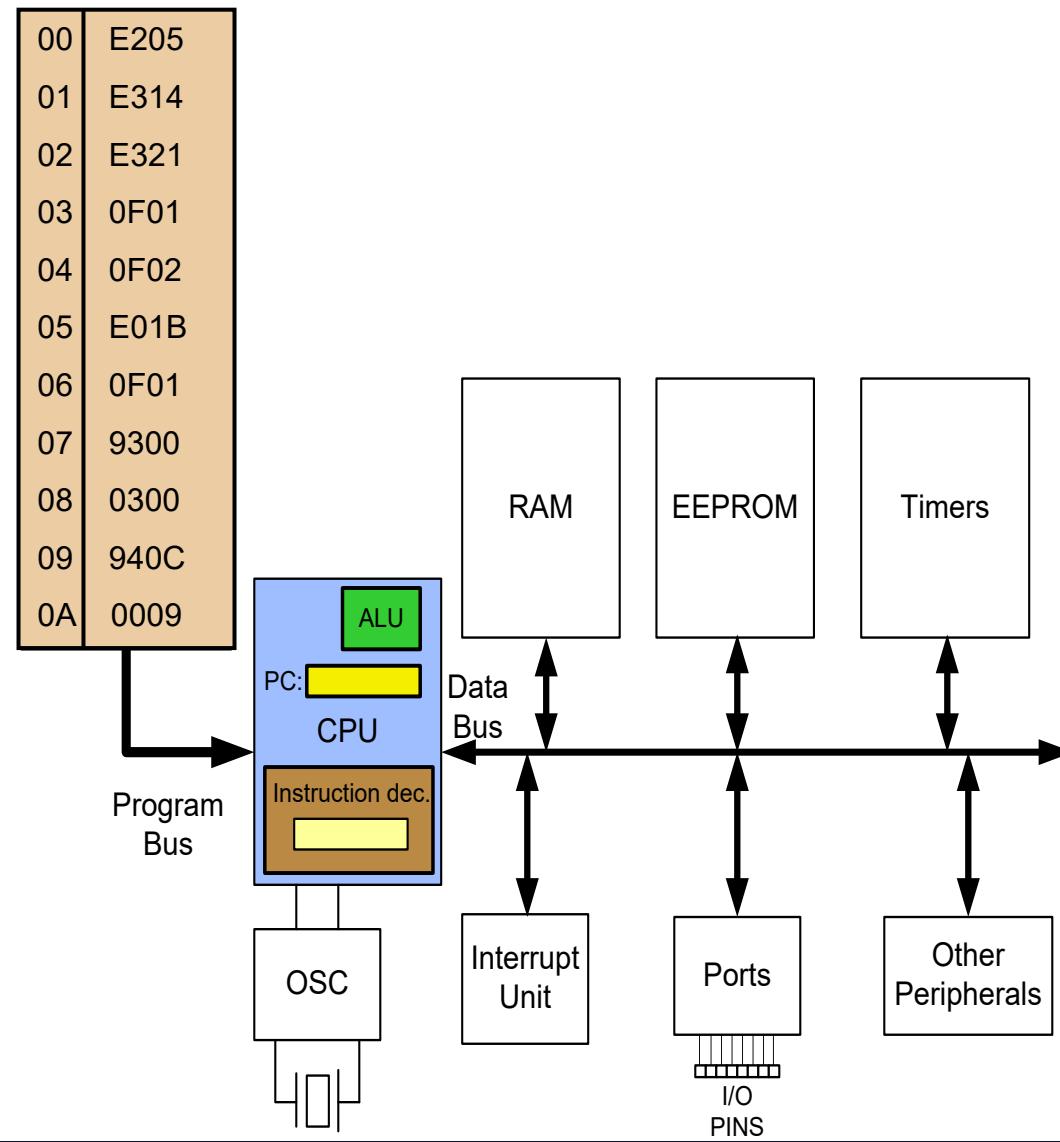
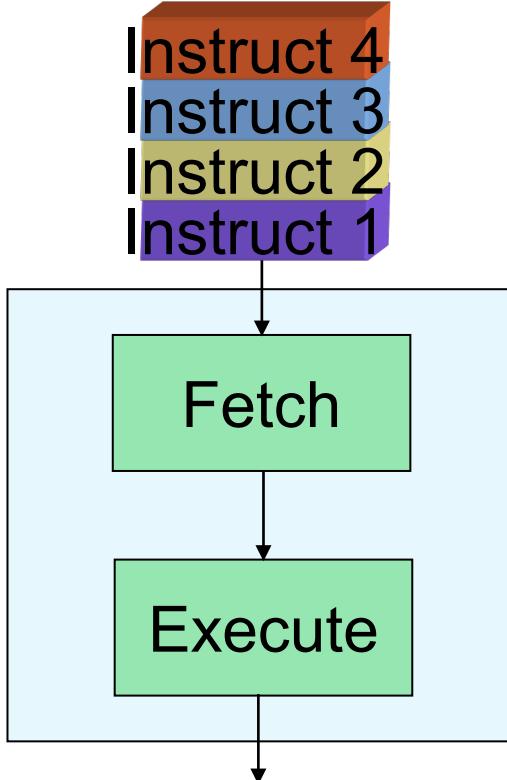


Flash memory and PC register



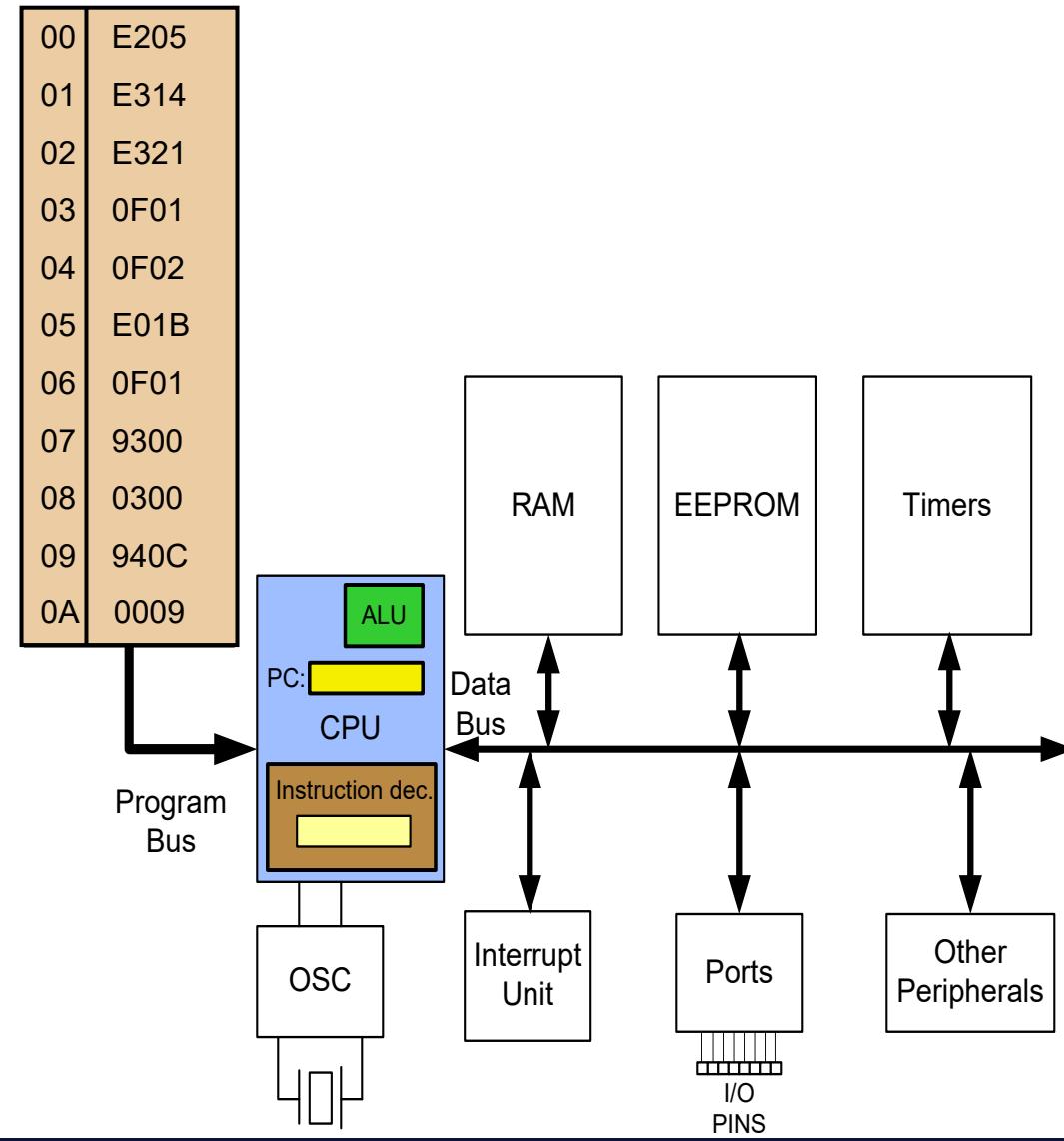
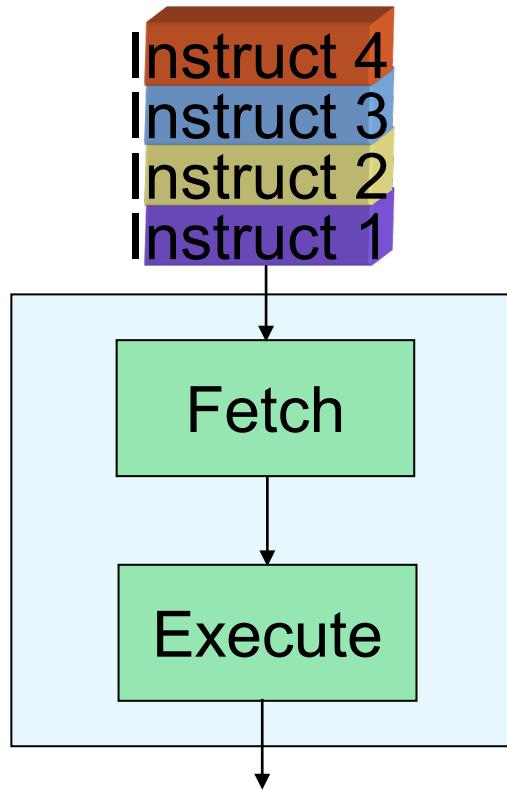
Fetch and execute

- Old Architectures



Pipelining

- Pipelining

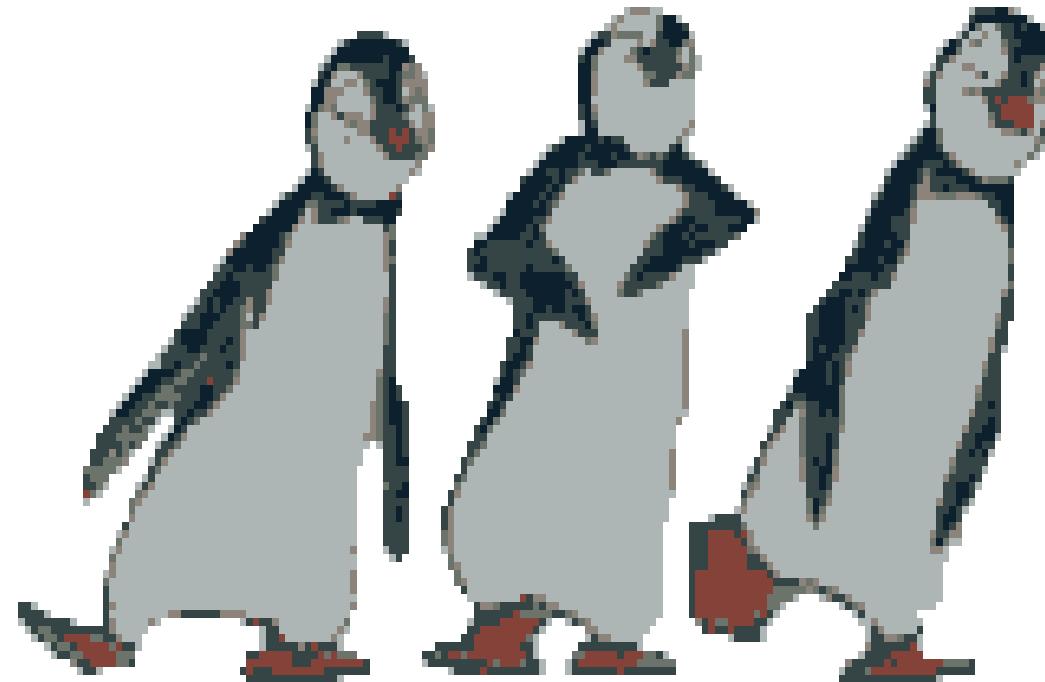


Programmet fra LAB1

```
;***** MSYS, LAB1 *****
;***** Henning Hargaard *****
;***** 14.august 2015 *****
;***** *****
;***** INITIERING *****
    LDI R16,HIGH(RAMEND) ;Initialize Stack Pointer
    OUT SPH,R16
    LDI R16,LOW(RAMEND)
    OUT SPL,R16
    SER R16                ;PORTB = Outputs
    OUT DDRB,R16

;***** PROGRAM-LOOP *****
    CLR R16
LOOP:
    LDI R17,9              ;R17 = 9
    ADD R16,R17              ;R16 = R16 + R17
    CALL DISP_AND_DELAY ;Display R16
    JMP LOOP               ;Jump to "LOOP"
;***** *****
;***** DISPLAY R16 *****
;***** AND DELAY *****
DISP_AND_DELAY:
    MOV R17,R16
    OUT PORTB,R17
    CLR R17
    CLR R18
    LDI R19,100
AGAIN:
    DEC R17
    BRNE AGAIN
    DEC R18
    BRNE AGAIN
    DEC R19
    BRNE AGAIN
    RET
;***** *****
```

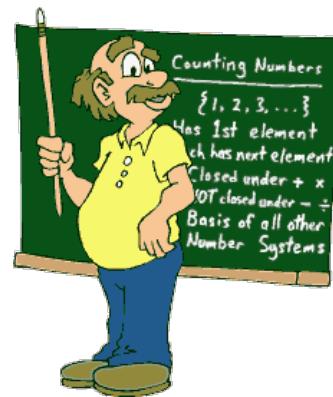
Slut



MSYS

Microcontroller Systems

Lektion 5: Assembly programmering



Slides from Henning Hargaard

Michael Alrøe

Instruktions-grupper

- Aritmetiske og logiske ("regne-instruktioner").
- Branch ("hop-instruktioner").
- Data transfer ("kopiering af data").
- Bit- og bit test –instruktioner.

LDI (Load Immediate)

Description:

Loads an 8 bit constant directly to register 16 to 31.

Operation:

(i) $Rd \leftarrow K$

Syntax:

(i) LDI Rd,K

Operands:

16 $\leq d \leq 31$, 0 $\leq K \leq 255$

Program Counter:

PC \leftarrow PC + 1

16-bit Opcode:

1110	KKKK	dddd	KKKK
------	------	------	------

Example:

```
CLR R31      ; Clear Z high byte
LDI R30,0xF0 ; Set Z low byte to $F0
LPM          ; Load constant from Program
              ; memory pointed to by Z
```

CLR (Clear register)

Description:

Clears a register. This instruction performs an Exclusive OR between a register and itself. This will clear all bits in the register.

Operation:

(i) $Rd \leftarrow Rd \oplus Rd$

Syntax:

(i) CLR Rd

Operands:

$0 \leq d \leq 31$

Program Counter:

$PC \leftarrow PC + 1$

16-bit Opcode: (see EOR Rd,Rd)

0010	01dd	dddd	dddd
------	------	------	------

Example:

```
    CLR  R18          ; clear r18
LOOP: INC  R18          ; increase r18
      ...
    CPI  R18,0x50    ; Compare r18 to $50
    BRNE LOOP
```

SER (Set all bits in register)

Description:

Loads \$FF directly to register Rd.

Operation:

- (i) $Rd \leftarrow \$FF$

Syntax:

- (i) SER Rd

Operands:

$16 \leq d \leq 31$

Program Counter:

$PC \leftarrow PC + 1$

16-bit Opcode:

1110	1111	dddd	1111
------	------	------	------

Example:

```
CLR R16          ; Clear r16
SER R17          ; Set r17
OUT PORTB,R16    ; Write zeros to Port B
NOP              ; Delay (do nothing)
OUT PORTB,R17    ; Write ones to Port B
```

MOV (Copy Register)

Description:

This instruction makes a copy of one register into another. The source register Rr is left unchanged, while the destination register Rd is loaded with a copy of Rr.

Operation:

- (i) $Rd \leftarrow Rr$

Syntax:

(i) $MOV\ Rd,Rr$

Operands:

$0 \leq d \leq 31, 0 \leq r \leq 31$

Program Counter:

$PC \leftarrow PC + 1$

16-bit Opcode:

0010	11rd	dddd	rrrr
------	------	------	------

Example:

```
MOV R16,R0          ; Copy r0 to r16
CALL CHECK          ; Call subroutine
...
CHECK: CPI R16,0x11 ; Compare r16 to $11
...
RET                 ; Return from subroutine
```

COM (Ones Complement)

Description:

This instruction performs a One's Complement of register Rd.

Operation:

(i) $Rd \leftarrow \$FF - Rd$

	Syntax:	Operands:
(i)	COM Rd	$0 \leq d \leq 31$

Program Counter:

$$PC \leftarrow PC + 1$$

16-bit Opcode:

1001	010d	dddd	0000
------	------	------	------

Example:

```
    COM R4          ; Take one's complement of r4
    BREQ ZERO      ; Branch if zero
    ...
ZERO: NOP          ; Branch destination (do nothing)
```

ADD (Add without Carry)

Description:

Adds two registers without the C Flag and places the result in the destination register Rd.

Operation:

(i) $Rd \leftarrow Rd + Rr$

Syntax:

(i) ADD Rd,Rr

Operands:

$0 \leq d \leq 31, 0 \leq r \leq 31$

Program Counter:

$PC \leftarrow PC + 1$

16-bit Opcode:

0000	11rd	dddd	rrrr
------	------	------	------

Example:

ADD R1,R2

; Add r2 to r1 ($r1=r1+r2$)

ADD R28,R28

; Add r28 to itself ($r28=r28+r28$)

SUB (Subtract without Carry)

Description:

Subtracts two registers and places the result in the destination register Rd.

Operation:

(i) $Rd \leftarrow Rd - Rr$

	Syntax:	Operands:	Program Counter:
(i)	SUB Rd,Rr	$0 \leq d \leq 31, 0 \leq r \leq 31$	$PC \leftarrow PC + 1$

16-bit Opcode:

0001	10rd	dddd	rrrr
------	------	------	------

Example:

```
SUB R13,R12          ; Subtract r12 from r13
BRNE NOTEQ           ; Branch if r12<>r13
...
NOTEQ: NOP           ; Branch destination (do nothing)
```

INC (Increment)

Description:

Adds one -1- to the contents of register Rd and places the result in the destination register Rd.

The C Flag in SREG is not affected by the operation, thus allowing the INC instruction to be used on a loop counter in multiple-precision computations.

When operating on unsigned numbers, only BREQ and BRNE branches can be expected to perform consistently. When operating on two's complement values, all signed branches are available.

Operation:

- (i) $Rd \leftarrow Rd + 1$

	Syntax:	Operands:	Program Counter:
(i)	INC Rd	$0 \leq d \leq 31$	$PC \leftarrow PC + 1$

16-bit Opcode:

1001	010d	dddd	0011
------	------	------	------

Example:

```
CLR R22          ; clear r22
LOOP: INC R22      ; increment r22
        CPI R22,0x4F ; Compare r22 to $4f
        BRNE LOOP    ; Branch if not equal
        NOP           ; Continue (do nothing)
```

DEC (Decrement)

Description:

Subtracts one -1- from the contents of register Rd and places the result in the destination register Rd.

The C Flag in SREG is not affected by the operation, thus allowing the DEC instruction to be used on a loop counter in multiple-precision computations.

When operating on unsigned values, only BREQ and BRNE branches can be expected to perform consistently. When operating on two's complement values, all signed branches are available.

Operation:

- (i) $Rd \leftarrow Rd - 1$

Syntax:	Operands:	Program Counter:
(i) DEC Rd	$0 \leq d \leq 31$	$PC \leftarrow PC + 1$

16-bit Opcode:

1001	010d	dddd	1010
------	------	------	------

Example:

```
LDI R17,16    ; Load constant in r17
LOOP: ADD R1,R2    ; Add r2 to r1
      DEC R17    ; Decrement r17
      BRNE LOOP    ; Branch if r17<>0
      NOP        ; Continue (do nothing)
```

Test ("socrative.com"): Room = AUQUIZ

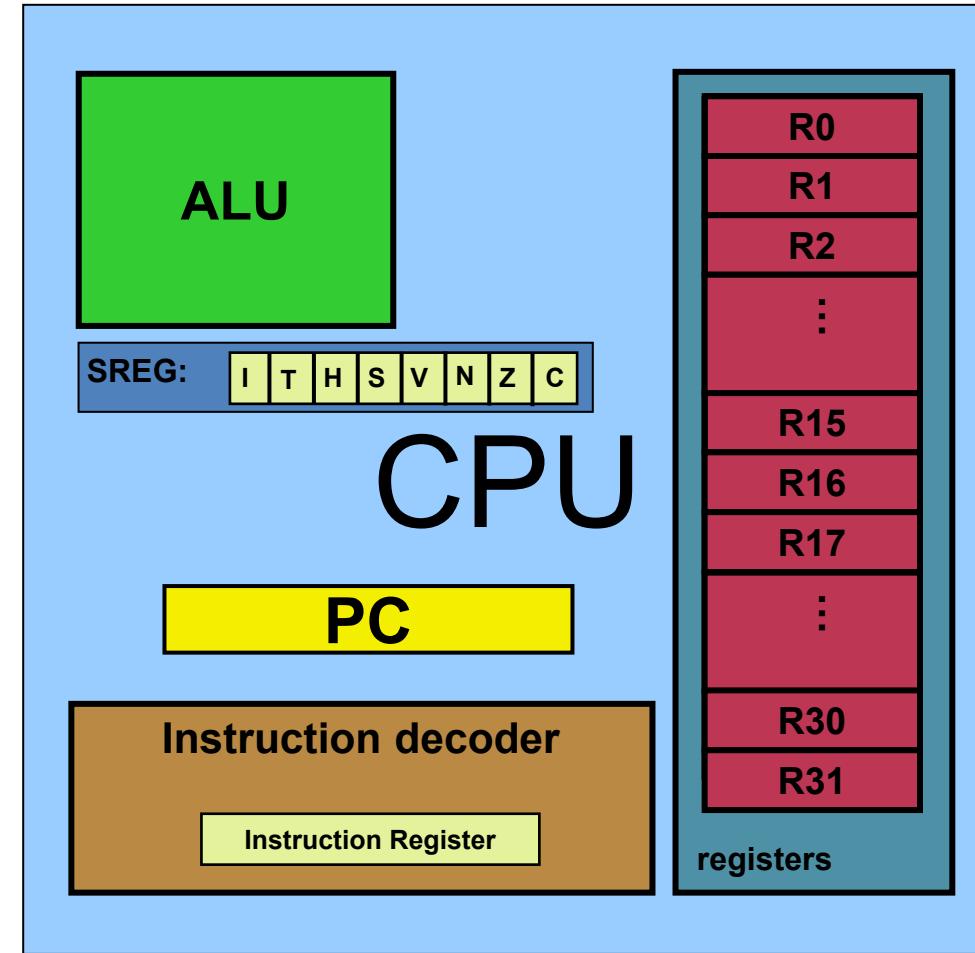
- Hvilket tal er i R20 efter følgende:

```
SER R20  
LDI R19,2  
LDI R20,200  
ADD R20,R19  
INC R20  
INC R20
```

- A: 200
- B: 202
- C: 203
- D: 204

AVR's CPU

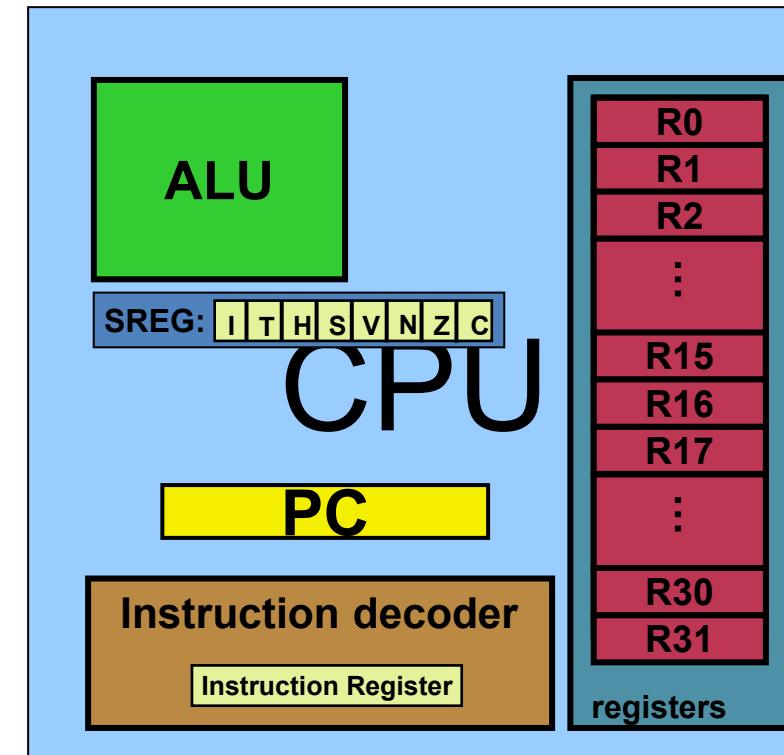
- AVR's CPU
 - ALU
 - 32 General Purpose registers (R0 to R31)
 - PC register
 - Instruction decoder



A simple program

- Write a program that calculates $19 + 95$

```
LDI R16, 19      ;R16 = 19  
LDI R20, 95      ;R20 = 95  
ADD R16, R20    ;R16 = R16 + R20
```



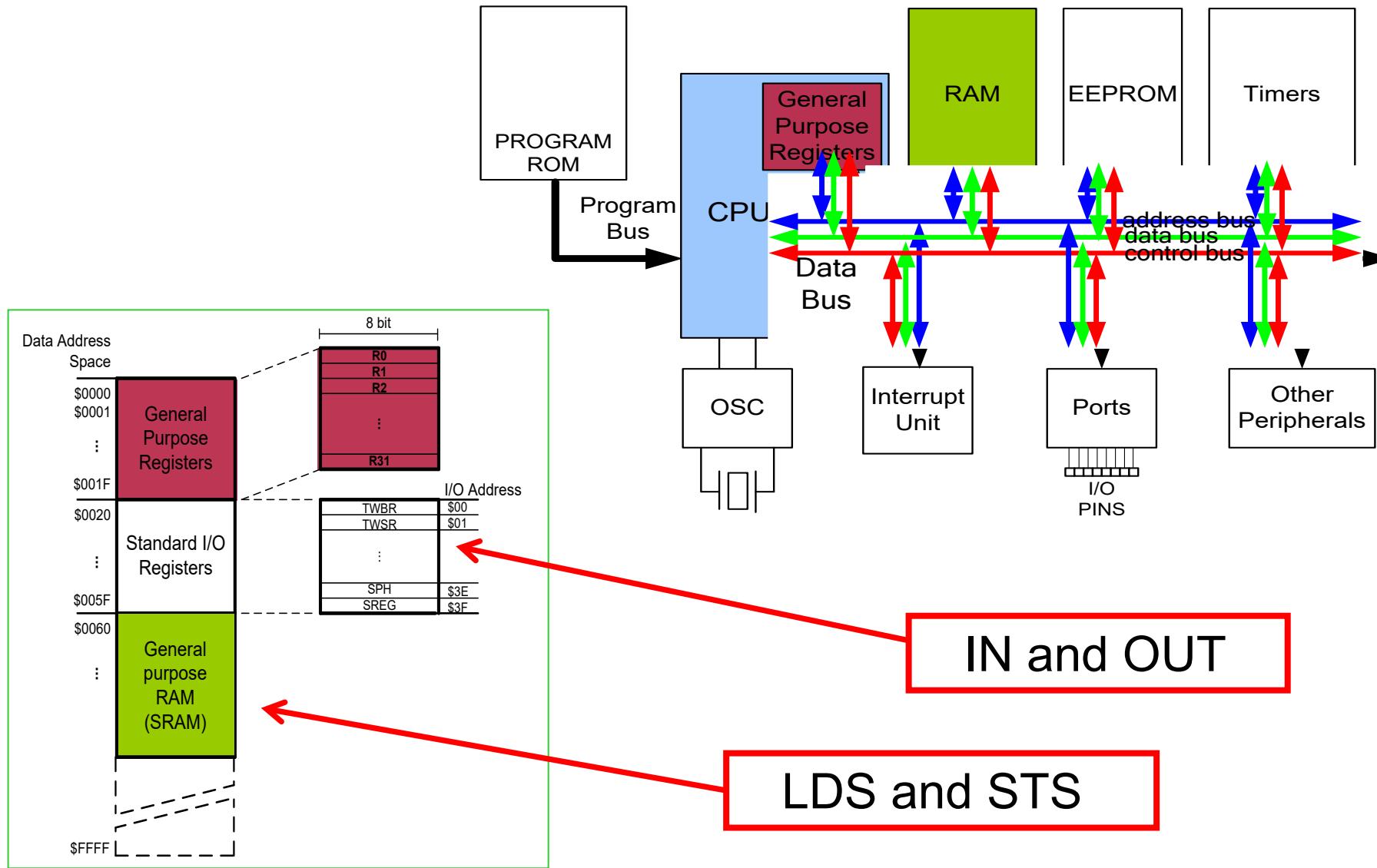
A simple program

- Write a program that calculates $19 + 95 + 5$

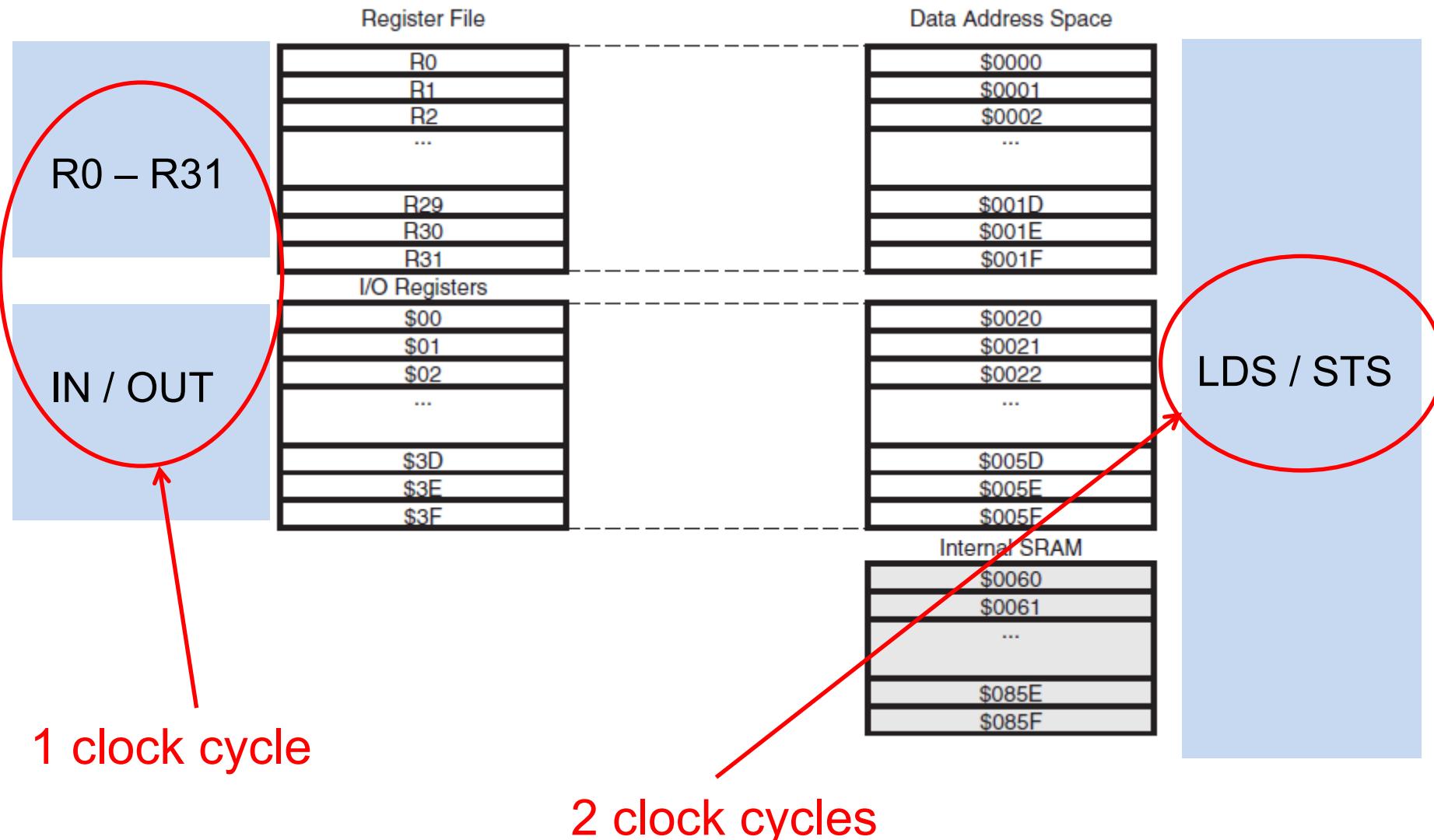
```
LDI    R16, 19      ;R16 = 19
LDI    R20, 95      ;R20 = 95
LDI    R21, 5       ;R21 = 5
ADD    R16, R20     ;R16 = R16 + R20
ADD    R16, R21     ;R16 = R16 + R21
```

```
LDI    R16, 19      ;R16 = 19
LDI    R20, 95      ;R20 = 95
ADD    R16, R20     ;R16 = R16 + R20
LDI    R20, 5       ;R20 = 5
ADD    R16, R20     ;R16 = R16 + R20
```

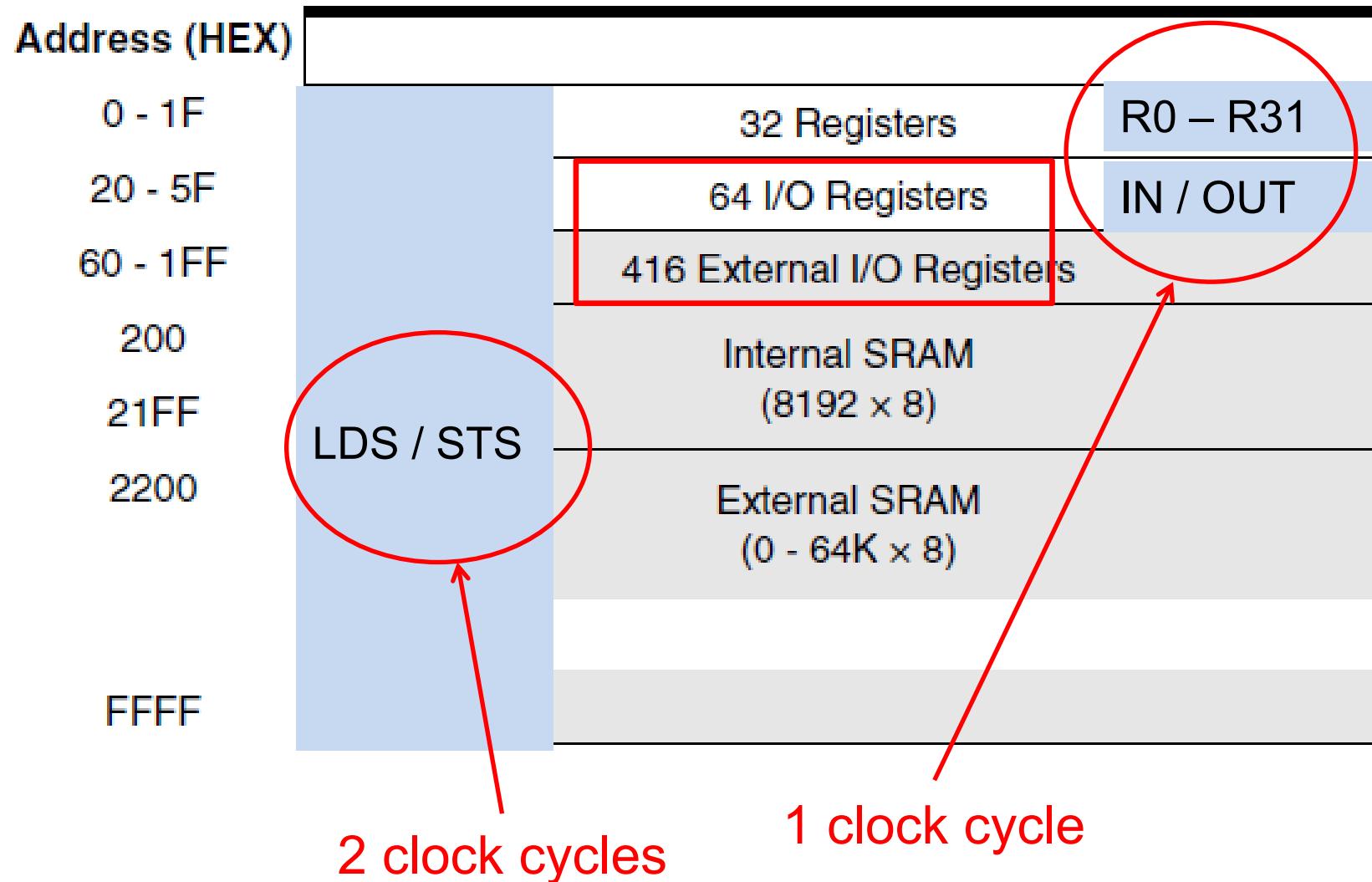
Data Address Space (Mega32)



Mega32 data memory



Mega2560 data memory



LDS (Load Direct from Data Space)

Description:

Loads one byte from the data space to a register. For parts with SRAM, the data space consists of the Register File, I/O memory and internal SRAM (and external SRAM if applicable). For parts without SRAM, the data space consists of the register file only. The EEPROM has a separate address space.

A 16-bit address must be supplied. Memory access is limited to the current data segment of 64K bytes. The LDS instruction uses the RAMPD Register to access memory above 64K bytes. To access another data segment in devices with more than 64K bytes data space, the RAMPD in register in the I/O area has to be changed.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

- (i) $Rd \leftarrow (k)$

Syntax:

- (i) LDS Rd,k

Operands:

$0 \leq d \leq 31, 0 \leq k \leq 65535$

Program Counter:

$PC \leftarrow PC + 2$

32-bit Opcode:

1001	000d	dddd	0000
kkkk	kkkk	kkkk	kkkk

Example:

```
LDS R2,0xFF00      ; Load r2 with the contents of data space location $FF00
ADD R2,R1          ; add r1 to r2
STS 0xFF00,R2      ; Write back
```

STS (Store Direct to Data Space)

Description:

Stores one byte from a Register to the data space. For parts with SRAM, the data space consists of the Register File, I/O memory and internal SRAM (and external SRAM if applicable). For parts without SRAM, the data space consists of the Register File only. The EEPROM has a separate address space.

A 16-bit address must be supplied. Memory access is limited to the current data segment of 64K bytes. The STS instruction uses the RAMPD Register to access memory above 64K bytes. To access another data segment in devices with more than 64K bytes data space, the RAMPD in register in the I/O area has to be changed.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

- (i) $(k) \leftarrow Rr$

Syntax:

- (i) STS k,Rr

Operands:

$0 \leq r \leq 31, 0 \leq k \leq 65535$

Program Counter:

$PC \leftarrow PC + 2$

32-bit Opcode:

1001	001d	dddd	0000
kkkk	kkkk	kkkk	kkkk

Example:

```
LDS R2,0xFF00      ; Load r2 with the contents of data space location $FF00
ADD R2,R1          ; add r1 to r2
STS 0xFF00,R2      ; Write back
```

IN (Load an I/O Location to Register)

Description:

Loads data from the I/O Space (Ports, Timers, Configuration Registers etc.) into register Rd in the Register File.

Operation:

- (i) $Rd \leftarrow I/O(A)$

Syntax:

- (i) IN Rd,A

Operands:

$0 \leq d \leq 31, 0 \leq A \leq 63$

Program Counter:

$PC \leftarrow PC + 1$

16-bit Opcode:

1011	0AA _d	dd _{dd}	AAA _A
------	------------------	------------------	------------------

Example:

```
IN R25,PINB ; Read Port B
CPI R25,4      ; Compare read value to constant
BREQ EXIT     ; Branch if r25=4
...
EXIT: NOP      ; Branch destination (do nothing)
```

OUT (Store Register to I/O Location)

Description:

Stores data from register Rr in the Register File to I/O Space (Ports, Timers, Configuration Registers etc.).

Operation:

(i) $I/O(A) \leftarrow Rr$

Syntax:

(i) OUT A,Rr

Operands:

$0 \leq r \leq 31, 0 \leq A \leq 63$

Program Counter:

$PC \leftarrow PC + 1$

16-bit Opcode:

1011	1AAr	rrrr	AAAA
------	------	------	------

Example:

```
CLR R16          ; Clear r16
SER R17          ; Set r17
OUT PORTB,R16    ; Write zeros to Port B
NOP              ; Wait (do nothing)
OUT PORTB,R17    ; Write ones to Port B
```

Assembler Directives .EQU (and .SET)

- *.EQU name = value*

– *Example:*

```
.EQU      COUNT = 0x25
LDI      R21, COUNT           ;R21 = 0x25
LDI      R22, COUNT + 3       ;R22 = 0x28
```

- *.SET name = value*

– *Example:*

```
.SET      COUNT = 0x25
LDI      R21, COUNT           ;R21 = 0x25
LDI      R22, COUNT + 3       ;R22 = 0x28
.SET      COUNT = 0x19
LDI      R21, COUNT           ;R21 = 0x19
```

Assembler Directives .INCLUDE

- .INCLUDE “*filename.ext*”

Table 2-6: Some of the common AVR's and their include files

MEGA	TINY	Special Purpose
Mega8	m8def.inc	Tiny11 tn11definc 90CAN32 can32definc

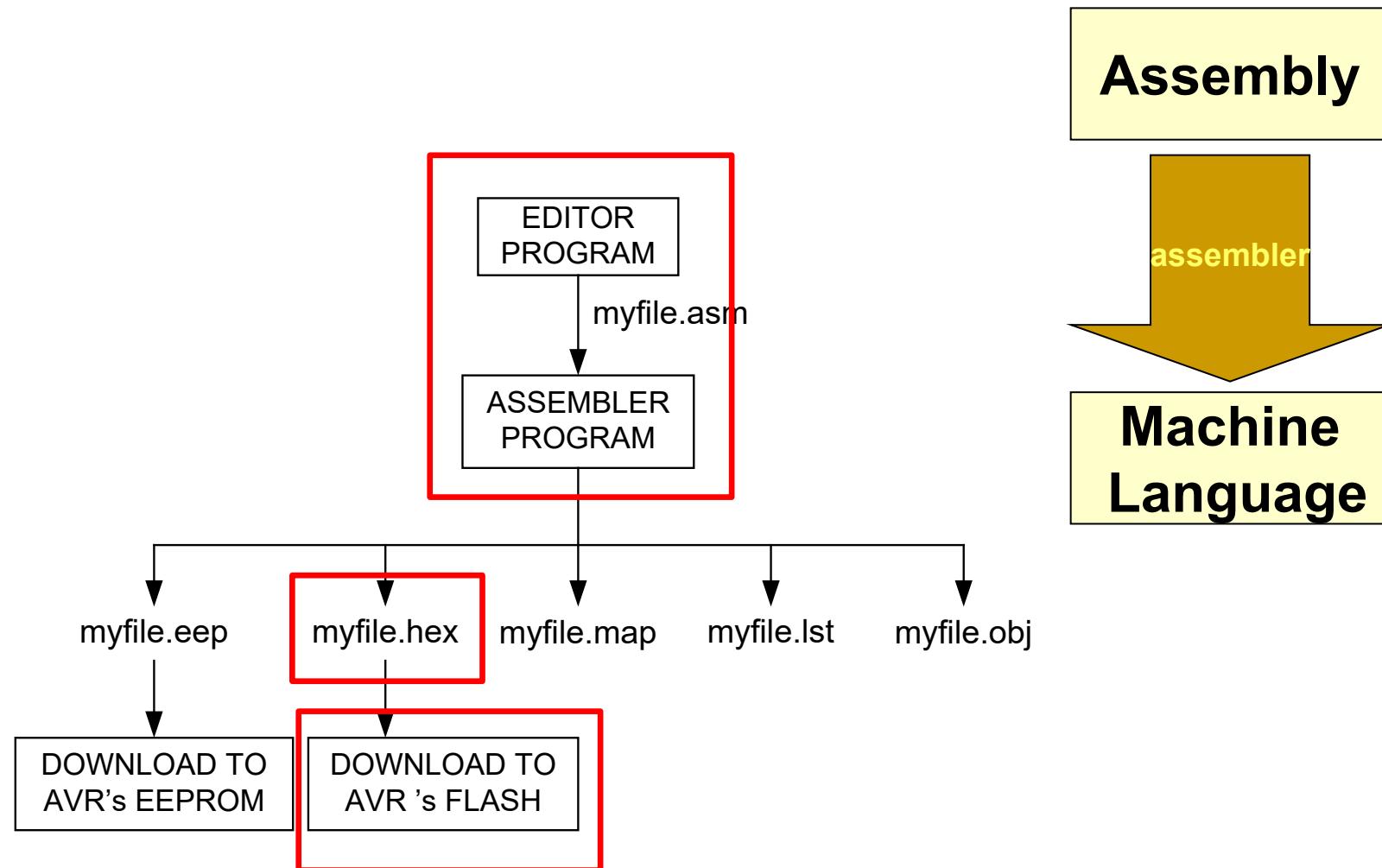
M32def.inc

```
.equ SREG = 0x3f
.equ SPL = 0x3d
.equ SPH = 0x3e
....
.equ INT_VECTORS_SIZE = 42 ; size in words
```

Program.asm

```
.INCLUDE "M32DEF.INC"
LDI R20, 10
OUT SPL, R20
```

Assembler

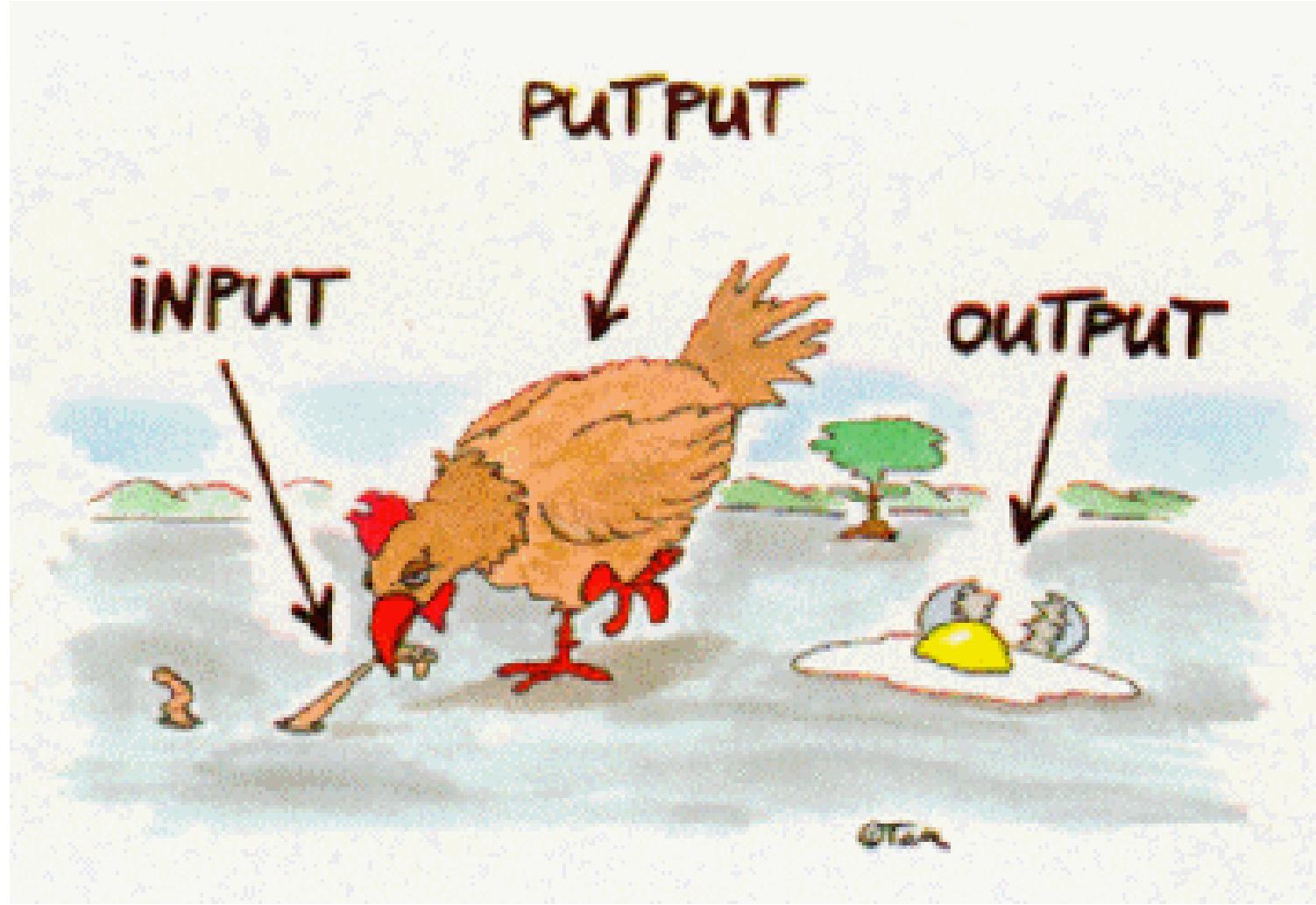


Programmet fra LAB1

```
;***** MSYS, LAB1 *****
;***** Henning Hargaard *****
;***** 14.august 2015 *****
;***** *****
;***** INITIERING *****
    LDI R16,HIGH(RAMEND) ;Initialize Stack Pointer
    OUT SPH,R16
    LDI R16,LOW(RAMEND)
    OUT SPL,R16
    SER R16                ;PORTB = Outputs
    OUT DDRB,R16

;***** PROGRAM-LOOP *****
    CLR R16
LOOP:
    LDI R17,9              ;R17 = 9
    ADD R16,R17              ;R16 = R16 + R17
    CALL DISP_AND_DELAY ;Display R16
    JMP LOOP               ;Jump to "LOOP"
;***** *****
;***** DISPLAY R16 *****
;***** AND DELAY *****
DISP_AND_DELAY:
    MOV R17,R16
    OUT PORTB,R17
    CLR R17
    CLR R18
    LDI R19,100
AGAIN:
    DEC R17
    BRNE AGAIN
    DEC R18
    BRNE AGAIN
    DEC R19
    BRNE AGAIN
    RET
;***** *****
```

Slut





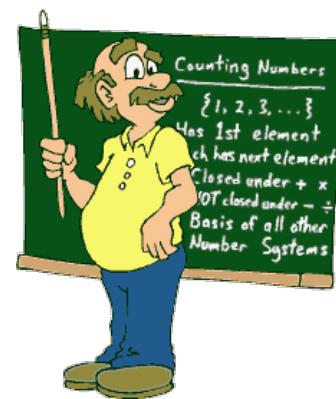
AARHUS
UNIVERSITY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

MSYS

Microcontroller Systems

Lektion 6: Status register og delays



Slides from Henning Hargaard

Michael Alrøe

Ubetinget "langt" Jump (JMP)

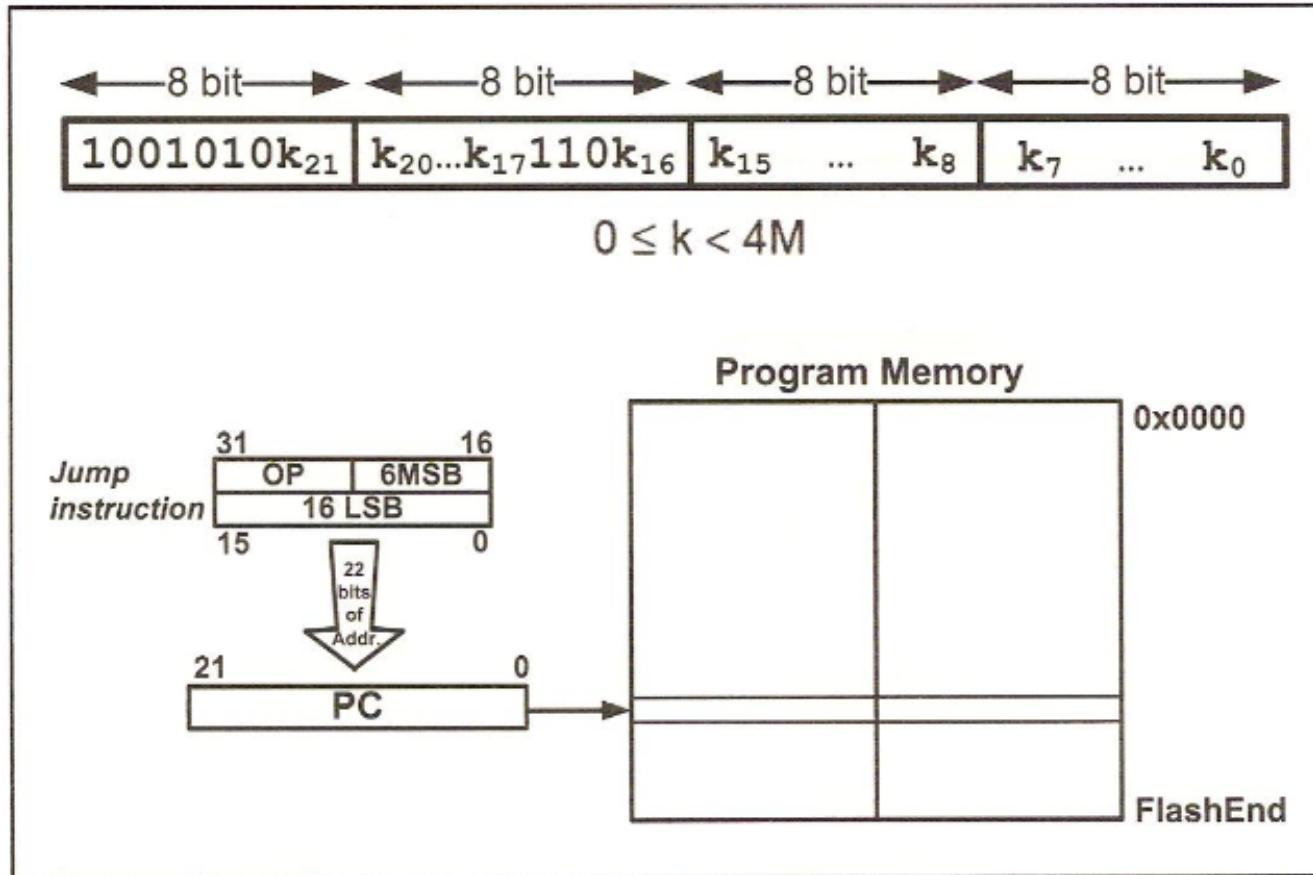


Figure 3-4. JMP Instruction

JMP IGEN

Ubetinget "kort" Jump (RJMP)

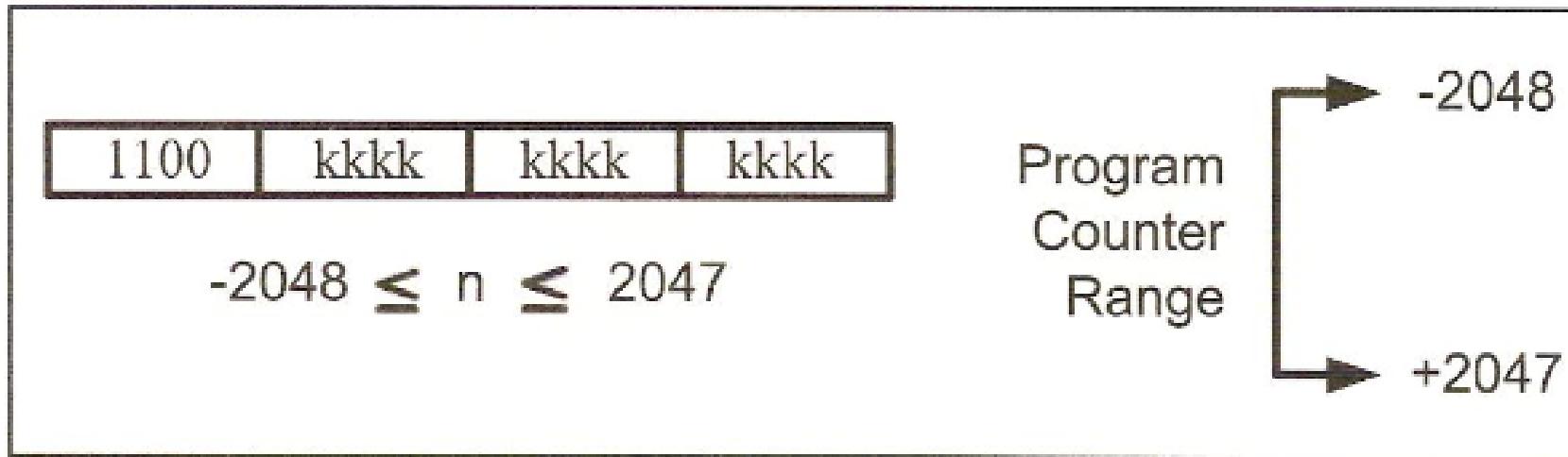
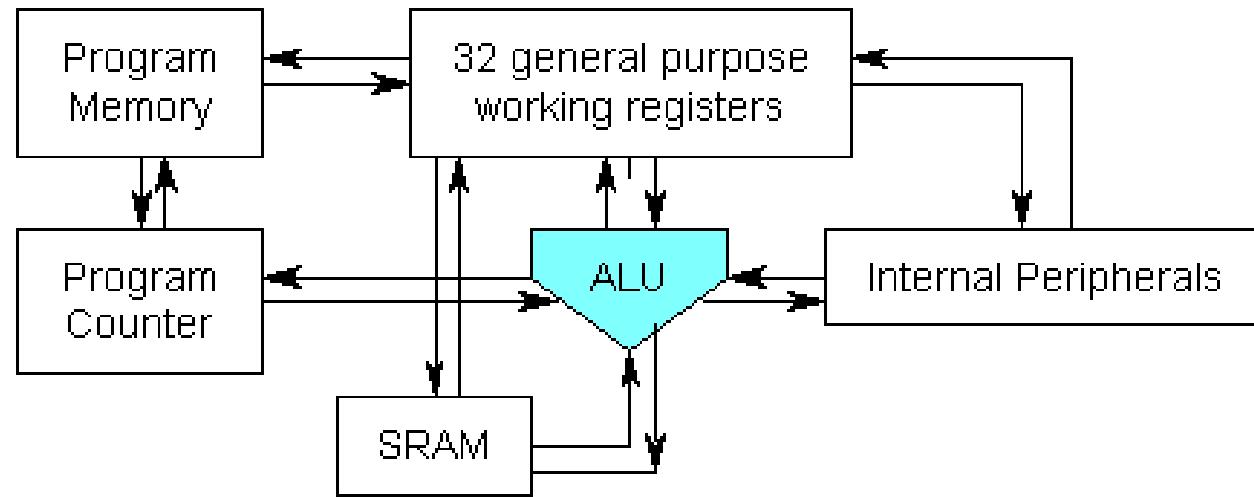


Figure 3-5. RJMP (Relative Jump) Instruction Address Range

RJMP IGEN

ALU



ALU'en er mikrocontrollerens "regnemaskine".

Hver gang vi udfører **en instruktion, der involverer ALU'en**, påvirkes nogen meget vigtige bits i status registeret (SREG) !

Hver bit i SREG kaldes "et **flag**".

Status register SREG

Bit	7	6	5	4	3	2	1	0	
Read/Write	R/W	SREG							
Initial Value	0	0	0	0	0	0	0	0	

- I-flag : Global interrupt enable (CLI og SEI).
- T-flag : Bit copy mellemklager (BLD og BST).
- H-flag : "Half carry" sættes ved "mente" fra bit 3 til bit 4 i en beregning. Bruges sjældent.
- S-flag : "Sign" kan anvendes ved beregning på negative tal. Afhænger af N- og V-flaget.
- V-flag : "2-komplement overflow" kan anvendes ved beregning på negative tal.
- N-flag : "Negative" indikerer et negativt resultat.
- **Z-flag : "Zero"** sættes, hvis resultatet bliver 0.
- **C-flag : "Carry"** sættes, hvis resultatet giver "mente".

Hvilke instruktioner bruger ALU'en ?

- Det er ikke alle instruktioner, der bruger regnemaskinen (ALU'en).
- Eksempelvis vil **MOV R12,R16** og **LDI R20,7** ikke påvirke statusregisteret (SREG).
- Eksempler på instruktioner, der påvirker SREG-flagene er :

INC R17

ADD R16,R17

DEC R4

Her påvirkes status-flagene

Table 2-4: Instructions That Affect Flag Bits

Instruction	C	Z	N	V	S	H
ADD	X	X	X	X	X	X
ADC	X	X	X	X	X	X
ADIW	X	X	X	X	X	
AND		X	X	X	X	
ANDI		X	X	X	X	
CBR		X	X	X	X	
CLR		X	X	X	X	
COM	X	X	X	X	X	
DEC		X	X	X	X	
EOR		X	X	X	X	
FMUL	X	X				
INC		X	X	X	X	
LSL	X	X	X	X		X
LSR	X	X	X	X		
OR		X	X	X	X	
ORI		X	X	X	X	
ROL	X	X	X	X		X
ROR	X	X	X	X		
SEN		1				
SEZ		1				
SUB	X	X	X	X	X	X
SUBI	X	X	X	X	X	X
TST		X	X	X	X	

Note: X can be 0 or 1. (See Chapter 5 for how to use these instructions.)

Betingede Jumps (bruger flagene)

Table 2-5: AVR Branch (Jump)
Instructions Using Flag Bits

Instruction	Action
BRLO	Branch if C = 1
BRSH	Branch if C = 0
BREQ	Branch if Z = 1
BRNE	Branch if Z = 0
BRMI	Branch if N = 1
BRPL	Branch if N = 0
BRVS	Branch if V = 1
BRVC	Branch if V = 0



Aritmetiske og logiske instruktioner(1)

Mnemonic	Operands	Description	Operation	Flags	Cycles
ADD	Rd,Rr	Add without Carry	$Rd = Rd + Rr$	Z,C,N,V,H,S	1
ADC	Rd,Rr	Add with Carry	$Rd = Rd + Rr + C$	Z,C,N,V,H,S	1
ADIW	Rd, K	Add Immediate To Word	$Rd+1 : Rd, K$	Z,C,N,V,S	2
SUB	Rd,Rr	Subtract without Carry	$Rd = Rd - Rr$	Z,C,N,V,H,S	1
SUBI	Rd,K8	Subtract Immediate	$Rd = Rd - K8$	Z,C,N,V,H,S	1
SBC	Rd,Rr	Subtract with Carry	$Rd = Rd - Rr - C$	Z,C,N,V,H,S	1
SBCI	Rd,K8	Subtract with Carry Immediate	$Rd = Rd - K8 - C$	Z,C,N,V,H,S	1
AND	Rd,Rr	Logical AND	$Rd = Rd \cdot Rr$	Z,N,V,S	1
ANDI	Rd,K8	Logical AND with Immediate	$Rd = Rd \cdot K8$	Z,N,V,S	1
OR	Rd,Rr	Logical OR	$Rd = Rd \vee Rr$	Z,N,V,S	1
ORI	Rd,K8	Logical OR with Immediate	$Rd = Rd \vee K8$	Z,N,V,S	1
EOR	Rd,Rr	Logical Exclusive OR	$Rd = Rd \text{ EOR } Rr$	Z,N,V,S	1
COM	Rd	One's Complement	$Rd = \$FF - Rd$	Z,C,N,V,S	1
NEG	Rd	Two's Complement	$Rd = \$00 - Rd$	Z,C,N,V,H,S	1
SBR	Rd,K8	Set Bit(s) in Register	$Rd = Rd \vee K8$	Z,C,N,V,S	1
CBR	Rd,K8	Clear Bit(s) in Register	$Rd = Rd \cdot (\$FF - K8)$	Z,C,N,V,S	1
INC	Rd	Increment Register	$Rd = Rd + 1$	Z,N,V,S	1
DEC	Rd	Decrement Register	$Rd = Rd - 1$	Z,N,V,S	1
TST	Rd	Test for Zero or Negative	$Rd = Rd \cdot Rd$	Z,C,N,V,S	1
CLR	Rd	Clear Register	$Rd = 0$	Z,C,N,V,S	1
SER	Rd	Set Register	$Rd = \$FF$	None	1

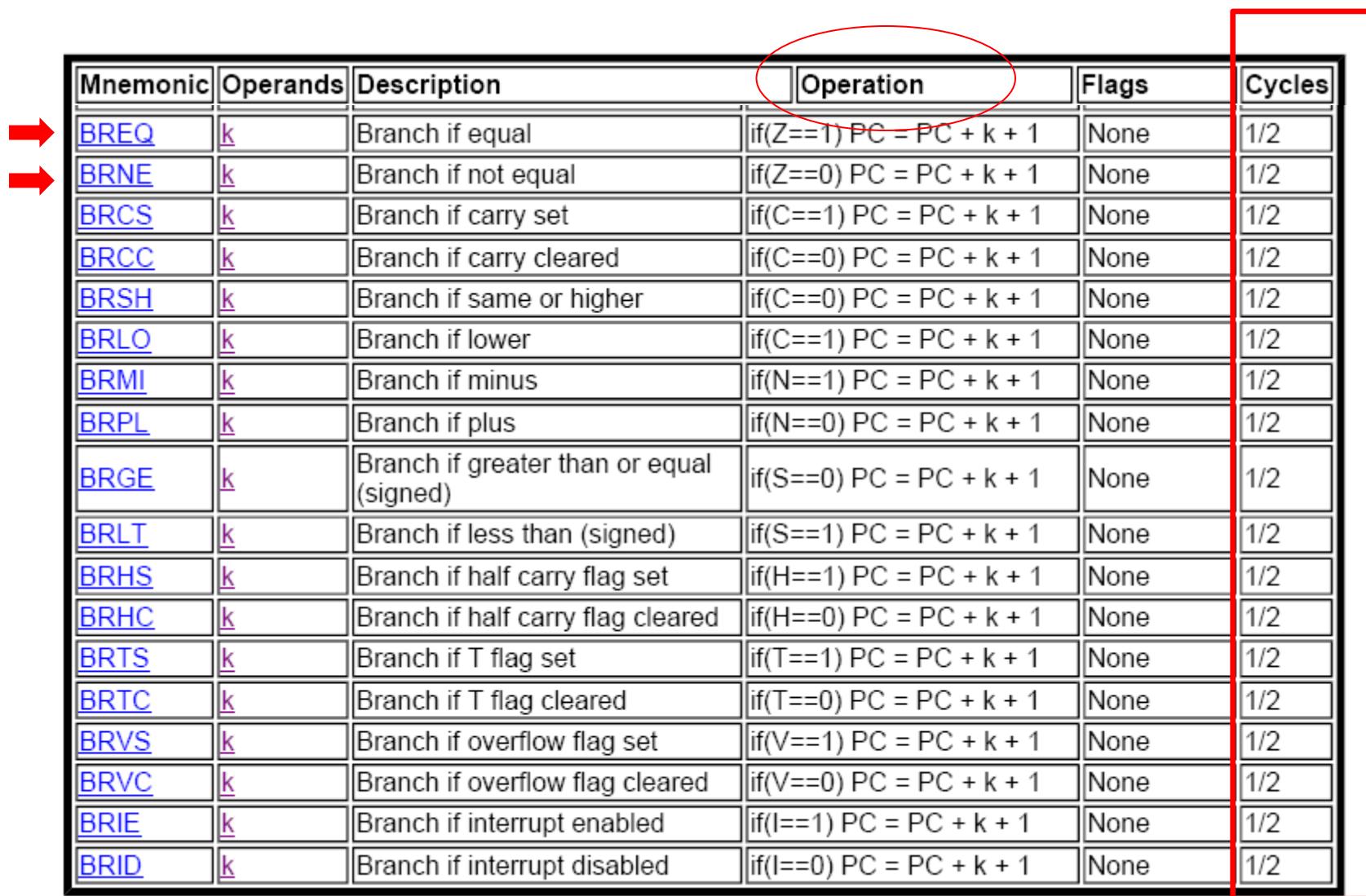
Aritmetiske og logiske instruktioner(2)

Mnemonic	Operands	Description	Operation	Flags	Cycles
SBIW	Rd,I,K6	Subtract Immediate from Word	Rdh:Rdl = Rdh:Rdl - K6	Z,C,N,V,S	2
MUL	Rd,Rr	Multiply Unsigned	R1:R0 = Rd * Rr	Z,C	2
MULS	Rd,Rr	Multiply Signed	R1:R0 = Rd * Rr	Z,C	2
MULSU	Rd,Rr	Multiply Signed with Unsigned	R1:R0 = Rd * Rr	Z,C	2
FMUL	Rd,Rr	Fractional Multiply Unsigned	R1:R0 = (Rd * Rr) << 1	Z,C	2
FMULS	Rd,Rr	Fractional Multiply Signed	R1:R0 = (Rd * Rr) << 1	Z,C	2
FMULSU	Rd,Rr	Fractional Multiply Signed with Unsigned	R1:R0 = (Rd * Rr) << 1	Z,C	2

Branch instruktioner (1)

Mnemonic	Operands	Description	Operation	Flags	Cycles
RJMP	k	Relative Jump	PC = PC + k + 1	None	2
IJMP	None	Indirect Jump to (Z)	PC = Z	None	2
JMP	k	Jump	PC = k	None	3
RCALL	k	Relative Call Subroutine	STACK = PC+1, PC = PC + k + 1	None	3/4*
ICALL	None	Indirect Call to (Z)	STACK = PC+1, PC = Z	None	3/4*
CALL	k	Call Subroutine	STACK = PC+2, PC = k	None	4/5*
RET	None	Subroutine Return	PC = STACK	None	4/5*
RETI	None	Interrupt Return	PC = STACK	I	4/5*
CPSE	Rd,Rr	Compare, Skip if equal	if (Rd == Rr) PC = PC 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd - Rr	Z,C,N,V,H,\$	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,H,\$	1
CPI	Rd,K8	Compare with Immediate	Rd - K	Z,C,N,V,H,\$	1
SBRC	Rr,b	Skip if bit in register cleared	if(Rr(b)==0) PC = PC + 2 or 3	None	1/2/3
SBRS	Rr,b	Skip if bit in register set	if(Rr(b)==1) PC = PC + 2 or 3	None	1/2/3
SBIC	P,b	Skip if bit in I/O register cleared	if(I/O(P,b)==0) PC = PC + 2 or 3	None	1/2/3
SBIS	P,b	Skip if bit in I/O register set	if(I/O(P,b)==1) PC = PC + 2 or 3	None	1/2/3
BRBC	s,k	Branch if Status flag cleared	if(SREG(s)==0) PC = PC + k + 1	None	1/2
BRBS	s,k	Branch if Status flag set	if(SREG(s)==1) PC = PC + k + 1	None	1/2

Branch instruktioner (2)



The diagram shows a table of branch instructions. Two rows are highlighted with red arrows pointing to them: BREQ and BRNE. A red circle highlights the 'Operation' column header. A red box highlights the last three columns: 'Flags' and 'Cycles'. The table has 18 rows, each representing a different branch instruction with its mnemonic, operands, description, operation, flags, and cycles.

Mnemonic	Operands	Description	Operation	Flags	Cycles
BREQ	k	Branch if equal	if(Z==1) PC = PC + k + 1	None	1/2
BRNE	k	Branch if not equal	if(Z==0) PC = PC + k + 1	None	1/2
BRCS	k	Branch if carry set	if(C==1) PC = PC + k + 1	None	1/2
BRCC	k	Branch if carry cleared	if(C==0) PC = PC + k + 1	None	1/2
BRSH	k	Branch if same or higher	if(C==0) PC = PC + k + 1	None	1/2
BRLO	k	Branch if lower	if(C==1) PC = PC + k + 1	None	1/2
BRMI	k	Branch if minus	if(N==1) PC = PC + k + 1	None	1/2
BRPL	k	Branch if plus	if(N==0) PC = PC + k + 1	None	1/2
BRGE	k	Branch if greater than or equal (signed)	if(S==0) PC = PC + k + 1	None	1/2
BRLT	k	Branch if less than (signed)	if(S==1) PC = PC + k + 1	None	1/2
BRHS	k	Branch if half carry flag set	if(H==1) PC = PC + k + 1	None	1/2
BRHC	k	Branch if half carry flag cleared	if(H==0) PC = PC + k + 1	None	1/2
BRTS	k	Branch if T flag set	if(T==1) PC = PC + k + 1	None	1/2
BRTC	k	Branch if T flag cleared	if(T==0) PC = PC + k + 1	None	1/2
BRVS	k	Branch if overflow flag set	if(V==1) PC = PC + k + 1	None	1/2
BRVC	k	Branch if overflow flag cleared	if(V==0) PC = PC + k + 1	None	1/2
BRIE	k	Branch if interrupt enabled	if(I==1) PC = PC + k + 1	None	1/2
BRID	k	Branch if interrupt disabled	if(I==0) PC = PC + k + 1	None	1/2

Data transfer instruktioner (1)

Mnemonic	Operands	Description	Operation	Flags	Cycles
MOV	Rd,Rr	Copy register	Rd = Rr	None	1
MOVW	Rd,Rr	Copy register pair	Rd+1:Rd = Rr+1:Rr, r,d even	None	1
LDI	Rd,K8	Load Immediate	Rd = K	None	1
LDS	Rd,k	Load Direct	Rd = (k)	None	2*
LD	Rd,X	Load Indirect	Rd = (X)	None	2*
LD	Rd,X+	Load Indirect and Post-Increment	Rd = (X), X=X+1	None	2*
LD	Rd,-X	Load Indirect and Pre-Decrement	X=X-1, Rd = (X)	None	2*
LD	Rd,Y	Load Indirect	Rd = (Y)	None	2*
LD	Rd,Y+	Load Indirect and Post-Increment	Rd = (Y), Y=Y+1	None	2*
LD	Rd,-Y	Load Indirect and Pre-Decrement	Y=Y-1, Rd = (Y)	None	2*
LDD	Rd,Y+q	Load Indirect with displacement	Rd = (Y+q)	None	2*
LD	Rd,Z	Load Indirect	Rd = (Z)	None	2*
LD	Rd,Z+	Load Indirect and Post-Increment	Rd = (Z), Z=Z+1	None	2*
LD	Rd,-Z	Load Indirect and Pre-Decrement	Z=Z-1, Rd = (Z)	None	2*
LDD	Rd,Z+q	Load Indirect with displacement	Rd = (Z+q)	None	2*

Data transfer instruktioner (2)





Mnemonic	Operands	Description	Operation	Flags	Cycles
STS	k,Rr	Store Direct	(k) = Rr	None	2*
ST	X,Rr	Store Indirect	(X) = Rr	None	2*
ST	X+,Rr	Store Indirect and Post-Increment	(X) = Rr, X=X+1	None	2*
ST	-X,Rr	Store Indirect and Pre-Decrement	X=X-1, (X)=Rr	None	2*
ST	Y,Rr	Store Indirect	(Y) = Rr	None	2*
ST	Y+,Rr	Store Indirect and Post-Increment	(Y) = Rr, Y=Y+1	None	2
ST	-Y,Rr	Store Indirect and Pre-Decrement	Y=Y-1, (Y) = Rr	None	2
ST	Y+q,Rr	Store Indirect with displacement	(Y+q) = Rr	None	2
ST	Z,Rr	Store Indirect	(Z) = Rr	None	2
ST	Z+,Rr	Store Indirect and Post-Increment	(Z) = Rr, Z=Z+1	None	2
ST	-Z,Rr	Store Indirect and Pre-Decrement	Z=Z-1, (Z) = Rr	None	2
ST	Z+q,Rr	Store Indirect with displacement	(Z+q) = Rr	None	2
LPM	None	Load Program Memory	R0 = (Z)	None	3
LPM	Rd,Z	Load Program Memory	Rd = (Z)	None	3
LPM	Rd,Z+	Load Program Memory and Post-Increment	Rd = (Z), Z=Z+1	None	3
SPM	None	Store Program Memory	(Z) = R1:R0	None	-
IN	Rd,P	In Port	Rd = P	None	1
OUT	P,Rr	Out Port	P = Rr	None	1
PUSH	Rr	Push register on Stack	STACK = Rr	None	2
POP	Rd	Pop register from Stack	Rd = STACK	None	2

Bit- og Bit Test instruktioner (1)

Mnemonic	Operands	Description	Operation	Flags	Cycles
LSL	Rd	Logical shift left	$Rd(n+1)=Rd(n)$, $Rd(0)=0$, $C=Rd(7)$	Z,C,N,V,H,S	1
LSR	Rd	Logical shift right	$Rd(n)=Rd(n+1)$, $Rd(7)=0$, $C=Rd(0)$	Z,C,N,V,S	1
ROL	Rd	Rotate left through carry	$Rd(0)=C$, $Rd(n+1)=Rd(n)$, $C=Rd(7)$	Z,C,N,V,H,S	1
ROR	Rd	Rotate right through carry	$Rd(7)=C$, $Rd(n)=Rd(n+1)$, $C=Rd(0)$	Z,C,N,V,S	1
ASR	Rd	Arithmetic shift right	$Rd(n)=Rd(n+1)$, $n=0,\dots,6$	Z,C,N,V,S	1
SWAP	Rd	Swap nibbles	$Rd(3..0) = Rd(7..4)$, $Rd(7..4) = Rd(3..0)$	None	1
BSET	s	Set flag	$SREG(s) = 1$	$SREG(s)$	1
BCLR	s	Clear flag	$SREG(s) = 0$	$SREG(s)$	1
SBI	P,b	Set bit in I/O register	$I/O(P,b) = 1$	None	2
CBI	P,b	Clear bit in I/O register	$I/O(P,b) = 0$	None	2
BST	Rr,b	Bit store from register to T	$T = Rr(b)$	T	1
BLD	Rd,b	Bit load from register to T	$Rd(b) = T$	None	1

Bit- og Bit Test instruktioner (2)

Mnemonic	Operands	Description	Operation	Flags	Cycles
SEC	None	Set carry flag	$C = 1$	C	1
CLC	None	Clear carry flag	$C = 0$	C	1
SEN	None	Set negative flag	$N = 1$	N	1
CLN	None	Clear negative flag	$N = 0$	N	1
SEZ	None	Set zero flag	$Z = 1$	Z	1
CLZ	None	Clear zero flag	$Z = 0$	Z	1
SEI	None	Set interrupt flag	$I = 1$	I	1
CLI	None	Clear interrupt flag	$I = 0$	I	1
SES	None	Set signed flag	$S = 1$	S	1
CLN	None	Clear signed flag	$S = 0$	S	1
SEV	None	Set overflow flag	$V = 1$	V	1
CLV	None	Clear overflow flag	$V = 0$	V	1
SET	None	Set T-flag	$T = 1$	T	1
CLT	None	Clear T-flag	$T = 0$	T	1
SEH	None	Set half carry flag	$H = 1$	H	1
CLH	None	Clear half carry flag	$H = 0$	H	1
NOP	None	No operation	None	None	1
SLEEP	None	Sleep	See instruction manual	None	1
WDR	None	Watchdog Reset	See instruction manual	None	1



NOP (No Operation)

Description:

This instruction performs a single cycle No Operation.

Operation:

- (i) No

	Syntax:	Operands:	Program Counter:
(i)	NOP	None	$PC \leftarrow PC + 1$

16-bit Opcode:

0000	0000	0000	0000
------	------	------	------

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
CLR R16      ; Clear r16
SER R17      ; Set r17
OUT PORTB,R16 ; Write zeros to Port B
NOP          ; Wait (do nothing)
OUT PORTB,R17 ; Write ones to Port B
```

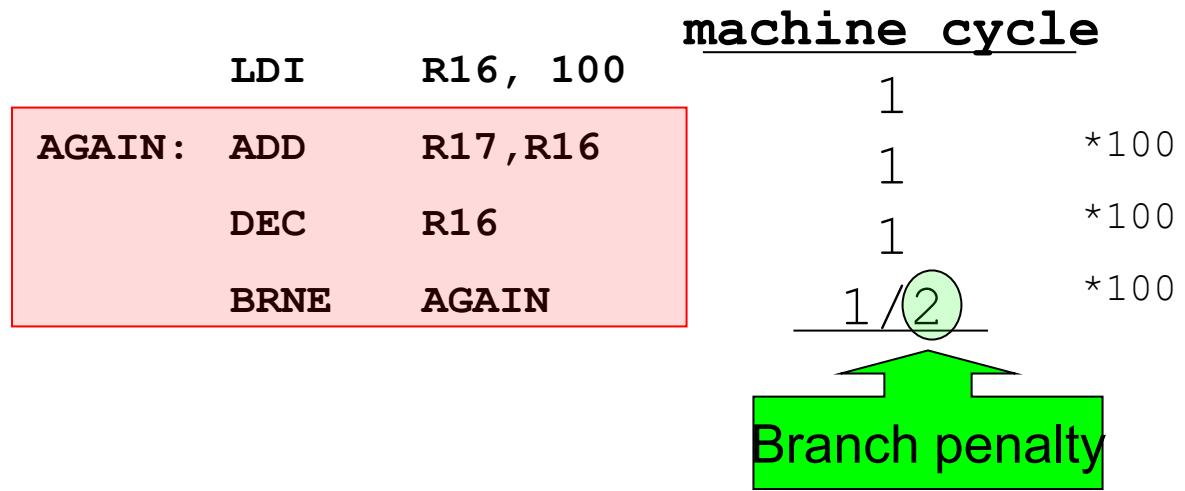
Words: 1 (2 bytes)

Cycles: 1

Time delay: Eksempel

<u>machine cycle</u>		
LDI	R16, 19	1
LDI	R20, 95	1
LDI	R21, 5	1
ADD	R16, R20	1
ADD	R16, R21	<u>1</u> 5

Time delay: Eksempel



Time delay: Eksempel

<u>machine cycle</u>			
LDI	R16, 50	1	
AGAIN:	NOP	1	*50
	NOP	1	*50
DEC	R16	1	*50
BRNE	AGAIN	1/2	*50

Time delay: Eksempel

	LDI	R17, 20	machine cycle
L1:	LDI	R16, 50	1 1 *20
L2:	NOP		1 *20 * 50
	NOP		1 *20 * 50
	DEC	R16	1 *20 * 50
	BRNE	L2	1/2 *20 * 50
	DEC	R17	1 *20
	BRNE	L1	1/2 *20

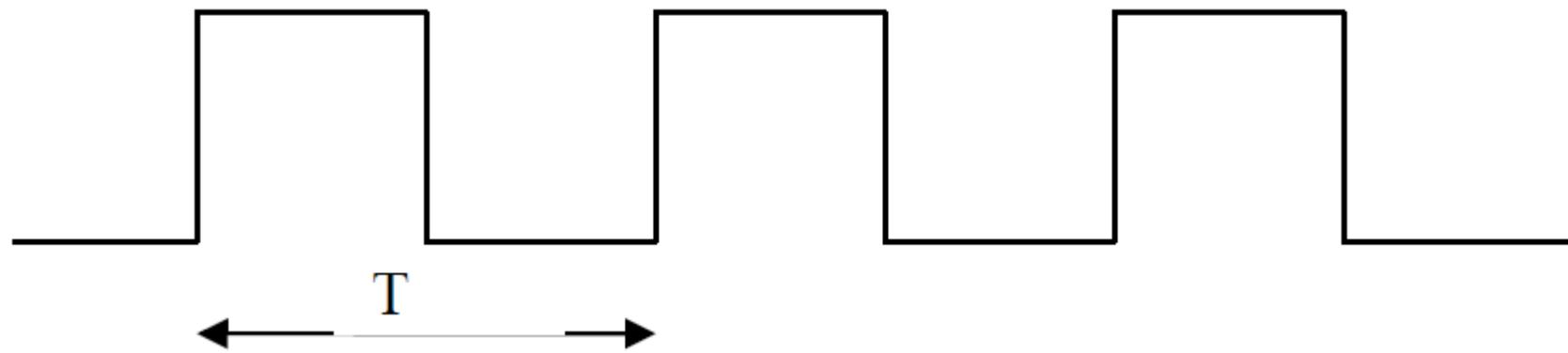
Kode fra LAB2

```
;***** DELAY *****
DELAY:
    CLR R17
    LDI R18,200 ;Stort tal = lang forsinkelse
AGAIN:
    DEC R17
    BRNE AGAIN
    DEC R18
    BRNE AGAIN
    RET
;*****
```

Simulatorens stopur (og cycle counter)

Name	Value
Program Counter	0x00000001
Stack Pointer	0x21FF
X Register	0x0000
Y Register	0x0000
Z Register	0x0000
Status Register	I T H S V N Z C
Cycle Counter	1
Frequency	16,000 MHz
Stop Watch	0,06 µs

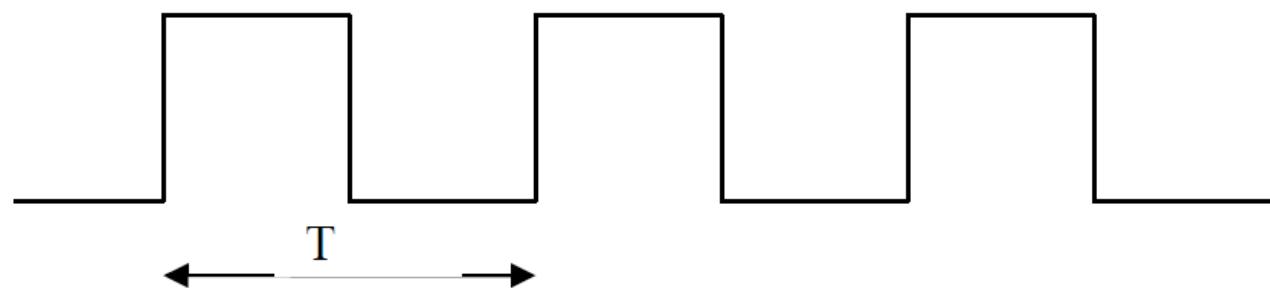
Oplæg til LAB3



Oplæg til LAB3

Tone	Frekvens	$T = 1/f$	$T/2$	$(T/2)/(4\mu s)$
C	523,25 Hz	1911 us	956 us	239
D	587,33 Hz	1792 us	851 us	213
E	659,26 Hz	1517 us	758 us	190
F	698,46 Hz	1432 us	716 us	179
G	783,99 Hz	1276 us	638 us	160
A	880,00 Hz	1136 us	568 us	142
H	987,77 Hz	1012 us	506 us	127
C	1046,50 Hz	956 us	478 us	120

Tabel 1: Frekvenserne + beregninger for en C dur skala.



LAB3

```
;***** PROGRAM LOOP *****
LDI R20,239 ;c      Spil C dur skalaen
CALL TONE
LDI R20,213 ;D
CALL TONE
LDI R20,190 ;E
CALL TONE
LDI R20,179 ;F
CALL TONE
LDI R20,160 ;G
CALL TONE
LDI R20,142 ;A
CALL TONE
LDI R20,127 ;H
CALL TONE
LDI R20,120 ;C
CALL TONE
HERE:
JMP HERE ;Bliv her "altid"
```

LAB3, del 2

```
;*****  
;***** SPILLER EN TONE *****  
;***** 250 HALVPERIODER *****  
;***** T/2 = (R20)*4 us *****  
;*****  
  
TONE:  
    LDI  R18,200    ;<-- Fjern denne instruktion, inden del 2 laves  
                  ;<-- HER MANGLER KODE (del 2)  
    CALL DELAY      ;Delay = (R18) * 4 mikrosekunder  
                  ;<-- HER MANGLER KODE (del 2)  
  
    RET             ;Retur til program loop
```

LAB3, del 1

```
;*****  
;*  DELAY (R18*4us)          *  
;*  Laver en tidsforsinkelser paa  *  
;* "Det der er i R18" gange 4 us  *  
;*****  
DELAY:  
    LDI  R17,xxx    ;<-- OPGAVE (del 1): Find xxx og skriv det her  
AGAIN:  
    DEC  R17  
    BRNE AGAIN  
    DEC  R18  
    BRNE DELAY  
    RET
```

Slut

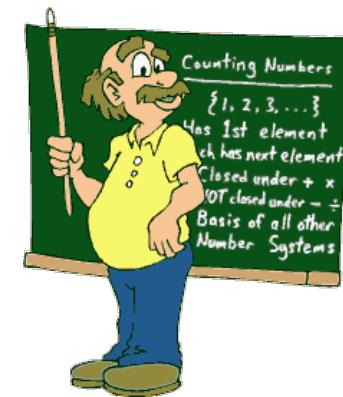


MSYS

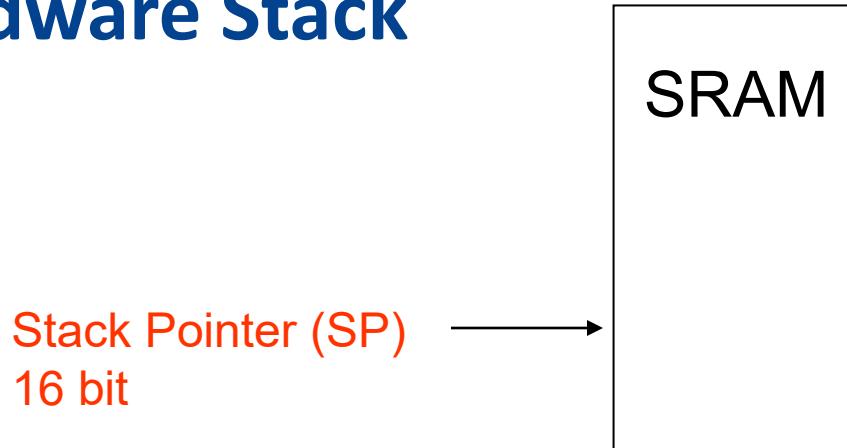
Microcontroller Systems

Lektion 7: Stack

Slides from Henning Hargaard



Hardware Stack



- Stack pointer'en er et vigtigt register, som **holder styr på, hvor "toppen af stacken" er** (d.v.s. næste ledige plads (adresse)).
- Hver gang data skrives til stacken, flyttes SP **automatisk tilbage** (dekrementeres).
- Hver gang data hentes fra stacken, flyttes SP **automatisk frem** (inkrementeres).

Stack pointer (SP) = SPH og SPL

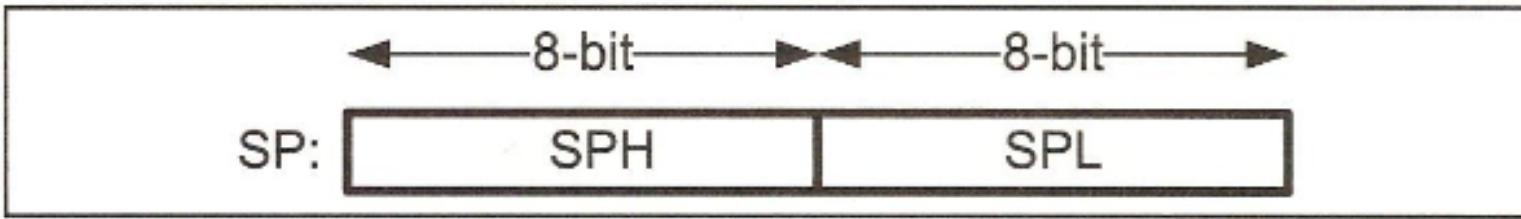


Figure 3-8. SP (Stack Pointer) in AVR

VIGTIGT:

Før brug skal stack pointeren initieres (sættes til at pege på "toppen af stacken") !

Det kan i Assembly gøres sådan:

```
LDI R16, HIGH(RAMEND)  
OUT SPH,R16  
LDI R16, LOW(RAMEND)  
OUT SPL,R16
```

Mega32:

RAMEND = 0x085F

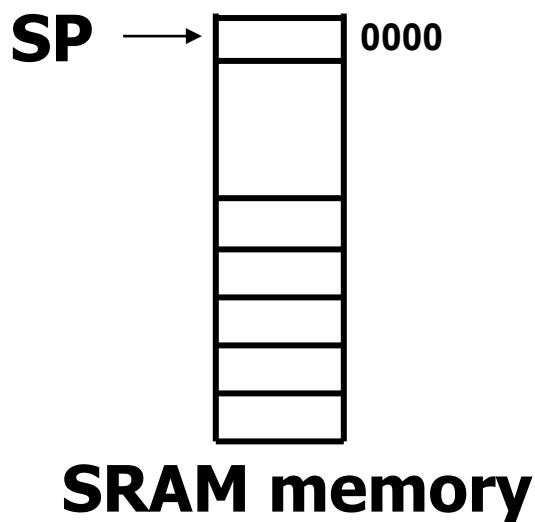
Mega2560:

RAMEND = 0x21FF

PUSH register / POP register

- Vi kan som (assembly-)programmører anvende stacken til at gemme data midlertidigt.
- ”PUSH register” (f.eks. **PUSH R17**) skriver register-indholdet til stacken (og **dekrementerer** automatisk SP).
- ”POP register” (f.eks. **POP R4**) læser data fra stacken til registeret (og **inkrementerer** automatisk SP).

Stack (PUSH og POP)



Address	Code
0000	ORG 0
0000	LDI R16, HIGH (RAMEND)
0001	OUT SPH, R16
0002	LDI R16, LOW (RAMEND)
0003	OUT SPL, R16
0004	LDI R20, 0x10
0005	LDI R21, 0x20
0006	LDI R22, 0x30
0007	PUSH \$10
0008	PUSH \$20
0009	PUSH \$30
000A	POP R21
000B	POP R0
000C	POP R20
000D	L1: RJMP L1

Test ("socrative.com"): Room = AUQUIZ

- Hvad indeholder R30 og R31 efter udførelse af følgende instruktioner:

LDI R30,30

LDI R31,31

PUSH R30

PUSH R31

POP R30

POP R31

A: $R30 = 30$ og $R31 = 31$

B: $R30 = 30$ og $R31 = 30$

C: $R30 = 31$ og $R31 = 31$

D: $R30 = 31$ og $R31 = 30$

CALL instruktionen

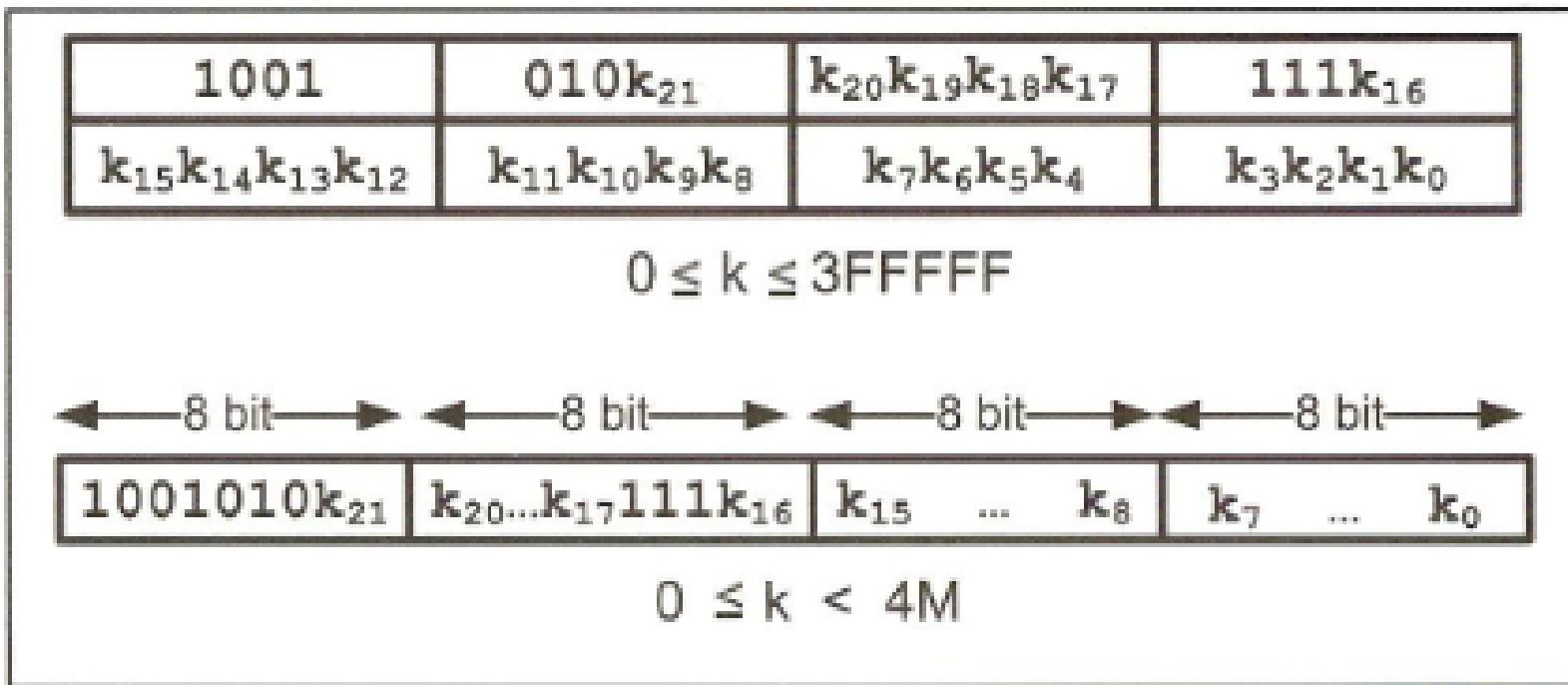


Figure 3-7. CALL Instruction Formation

Der findes også en "relative CALL", der fylder mindre.
Eksempel: **RCALL HUGO**

Subrutine-kald

MOV R4 ,R7

Adressen på næste instruktion gemmes på stacken. SP dekrementeres (med 2).

CALL check

MOV R0 ,R1

LDI R7 ,16

check: **MOV R0 ,R4**

INC R7

MOV R1 ,R3

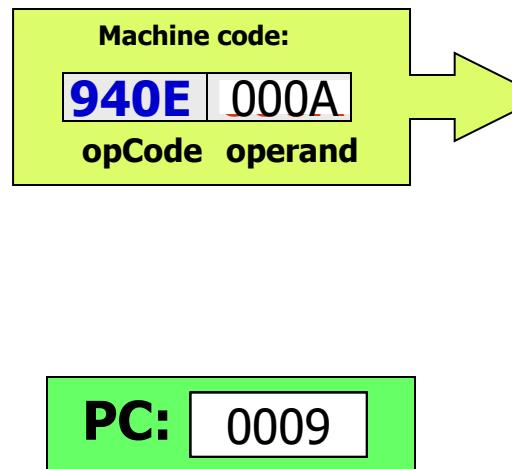
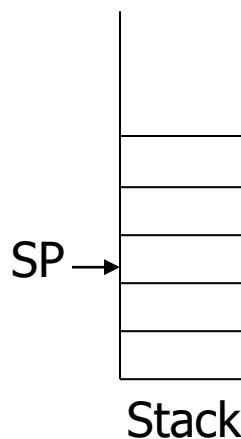
RET

Programtælleren (PC)
loades fra stacken.
SP inkrementeres (med 2).

- Ved subroutinekald anvendes stacken (automatisk) til at "finde tilbage" til instruktionen efter CALL :
Der gemmes "et bogmærke" på stacken.

Calling a Function

- To execute a call:
 - Address of the next instruction is saved
 - PC is loaded with the appropriate value



Address	Code
0000	LDI R16, HIGH(RAMEND)
0001	OUT SPH, R16
0002	LDI R16, LOW(RAMEND)
0003	OUT SPL, R16
0004	LDI R20, 15
0005	LDI R21, 5
0006	CALL FUNC_NAME
0008	INC R20
0009	L1: RJMP L1
000A	FUNC_NAME:
000A	ADD R20, R21
000B	SUBI R20, 3
000C	
000D	RET

Subroutine-kald i flere niveauer

```
MOV R4,R7
CALL check
MOV R0,R1
LDI R7,16
check:MOV R0,R4
INC R7
CALL s2
MOV R1,R3
RET
s2:INC R11
MOV R0,R7
RET
```

```
MOV R4,R7
CALL check
MOV R0,R1
LDI R7,16
check:MOV R0,R4
INC R7
CALL s2
MOV R1,R3
RET
s2:INC R11
MOV R0,R7
RET
```

- Ved kald i flere niveauer holder SP hele tiden styr på, hvor den aktuelle retur-adresse er på stacken (nemlig "toppen" af stacken).
- Pas på: Risiko for "stack overflow" !

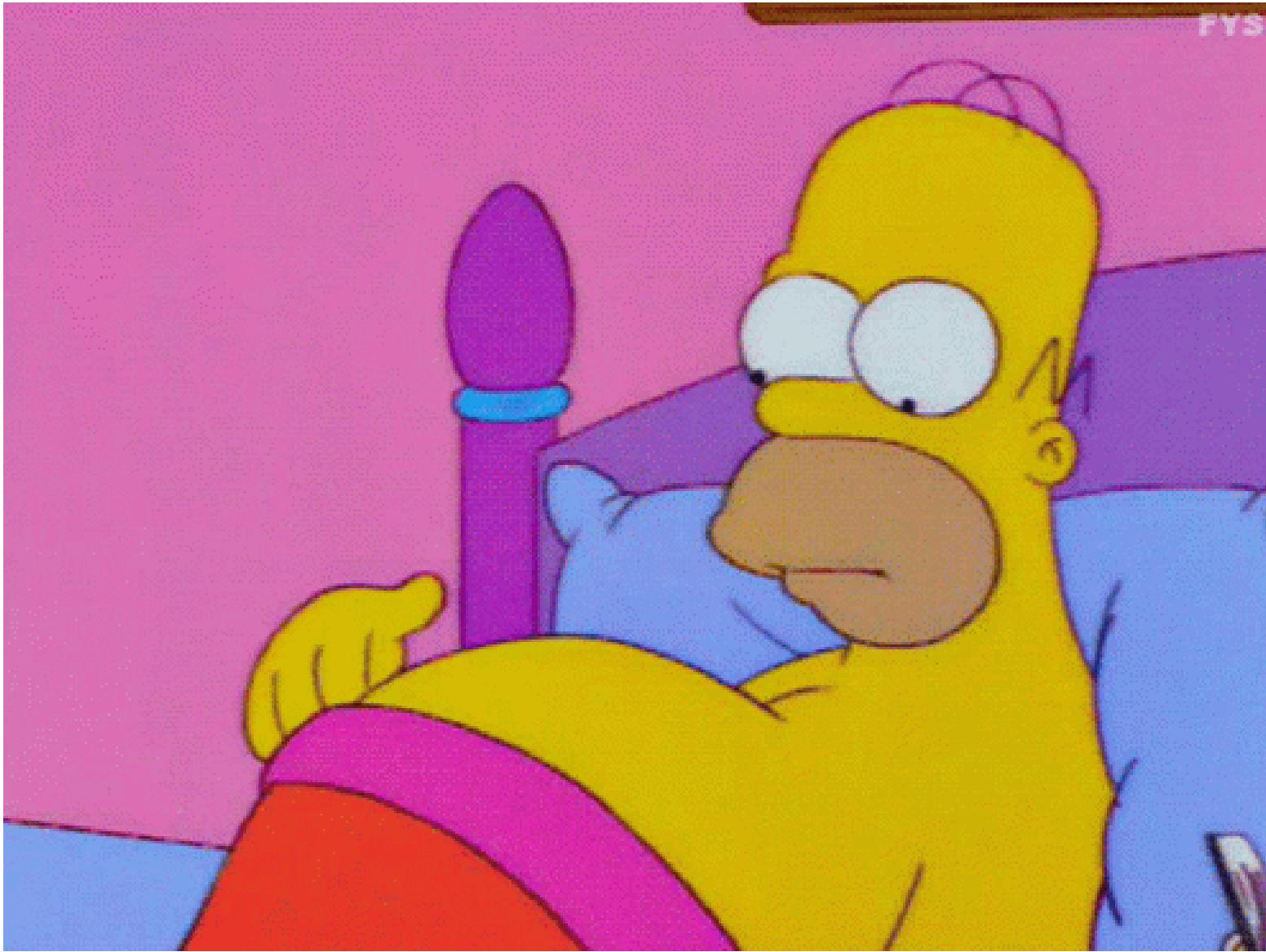
Typisk brug af CALL

```
.INCLUDE "M32DEF.INC" ;Modify for your chip

;MAIN program calling subroutines
.ORG 0
MAIN:    CALL SUBR_1
          CALL SUBR_2
          CALL SUBR_3
          CALL SUBR_4
HERE:    RJMP HERE      ;stay here
;-----end of MAIN
;
SUBR_1:   ....
          ....
          RET
;-----end of subroutine 1
;
SUBR_2:   ....
          ....
          RET
;-----end of subroutine 2
;
SUBR_3:   ....
          ....
          RET
;-----end of subroutine 3
;
SUBR_4:   ....
          ....
          RET
;-----end of subroutine 4
```

Figure 3-9. AVR Assembly Main Program That Calls Subroutines

Slut på lektion 7

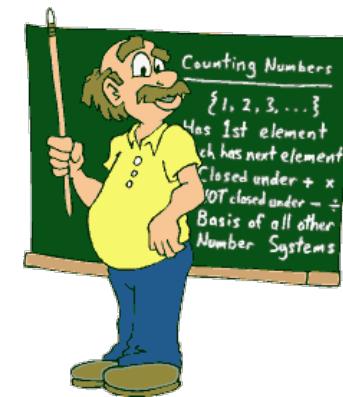


MSYS

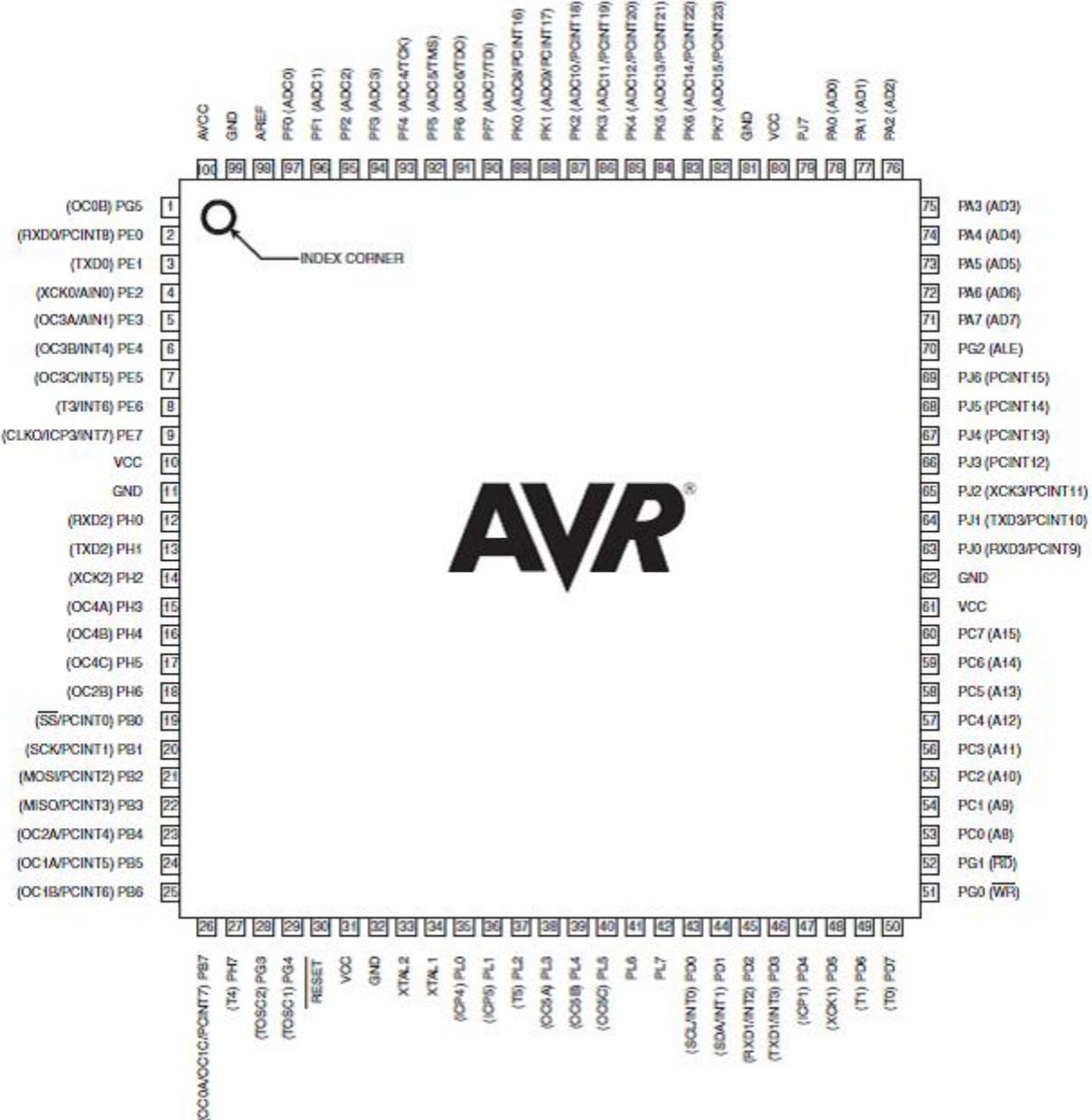
Microcontroller Systems

Lektion 8: Digitale Porte

Slides from Henning Hargaard



Mega2560 Pinout



AVR®

Forskellige AVR controllers Porte

Table 4-1: Number of Ports in Some AVR Family Members

Pins	8-pin	28-pin	40-pin	64-pin	100-pin
Chip	ATtiny25/45/85	ATmega8/48/88	ATmega32/16	ATmega64/128	ATmega1280
Port A			X	X	X
Port B	6 bits	X	X	X	X
Port C		7 bits	X	X	X
Port D		X	X	X	X
Port E				X	X
Port F				X	X
Port G				5 bits	6 bits
Port H					X
Port J					X
Port K					X
Port L					X

Mega2560

Note: X indicates that the port is available.

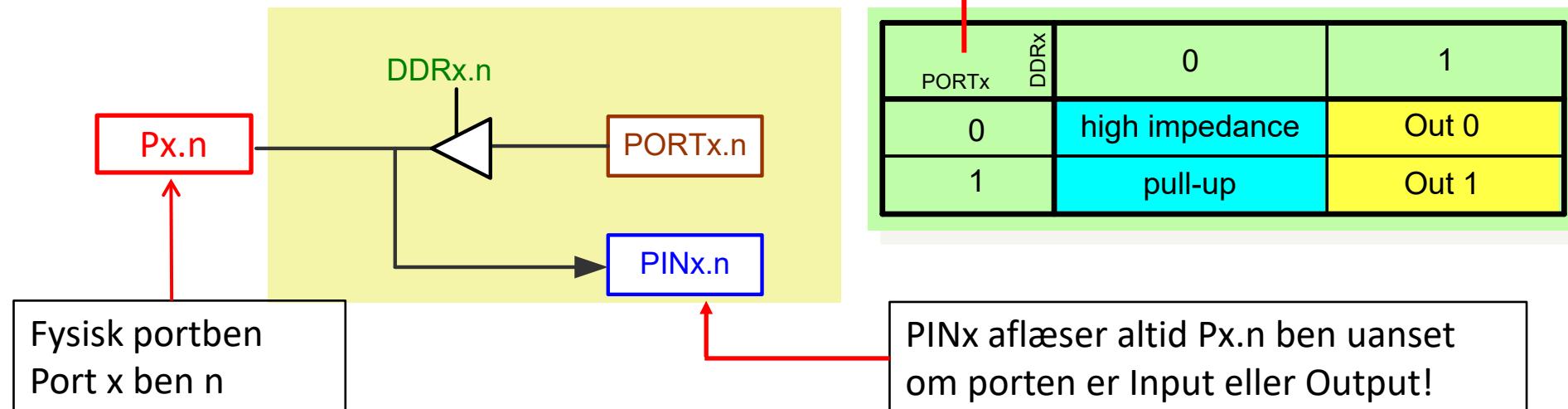
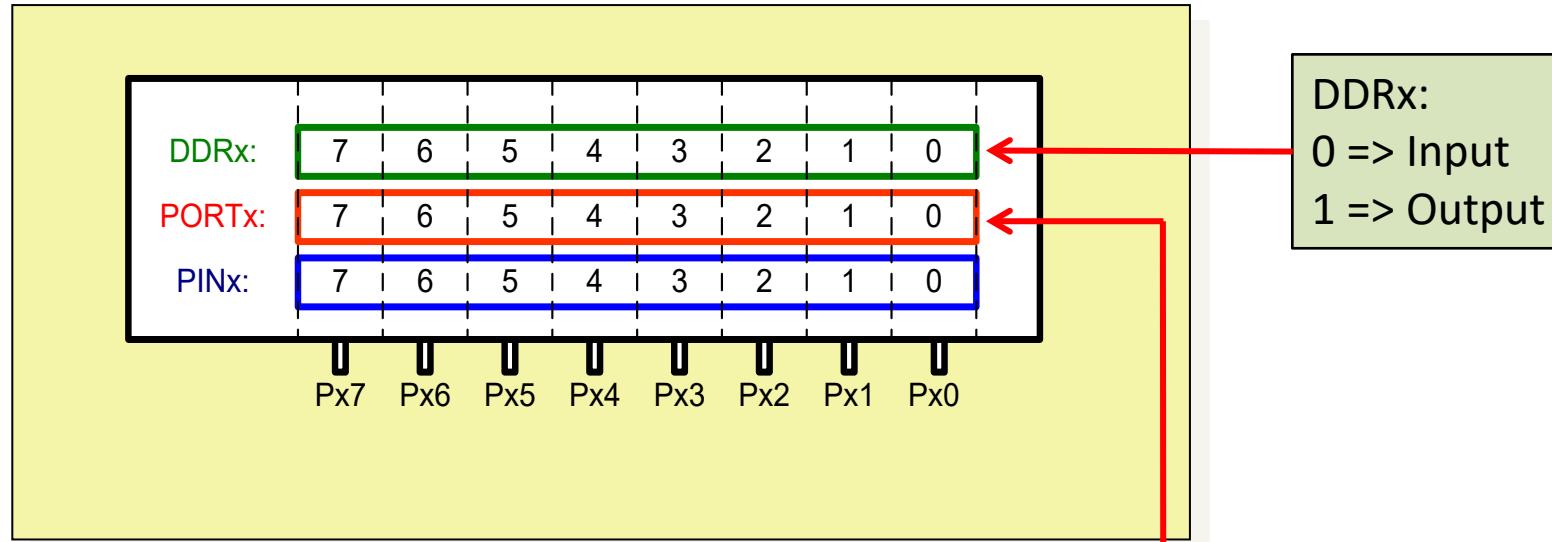
86 GPIO ben !!!

Digitale porte og registre

- Each of the AVR Digital I/O ports (Px) is associated with 3 I/O registers:
 - A Data Direction Register (**DDRx**)
 - A Pin Register (**PINx**)
 - A Port Register (**PORTx**)

‘x’ is A, B, C, etc.

Digitale porte og registre



DDR-registrene

- DDR-registrene opsætter de enkelte ben på en port til UD- eller INDGANGE (Inputs / Outputs).
- Et ben sat op til en UDGANG vil senere kunne bruges til at sende spænding UD på (0 eller 5 volt).
- Et ben sat op til en INDGANG vil senere kunne aflæses af programmet (undersøge, om der er 0 eller 5 volt på benet).
- Man skriver typisk kun til DDR-registrene i programmets start (initiering).
- Et ben vælges til **UDGANG** ved at skrive et **1** på den tilsvarende bitplads i DDR-registeret.
- Et ben vælges til **INDGANG** ved at skrive et **0** på den tilsvarende bitplads i DDR-registeret (dette er default).

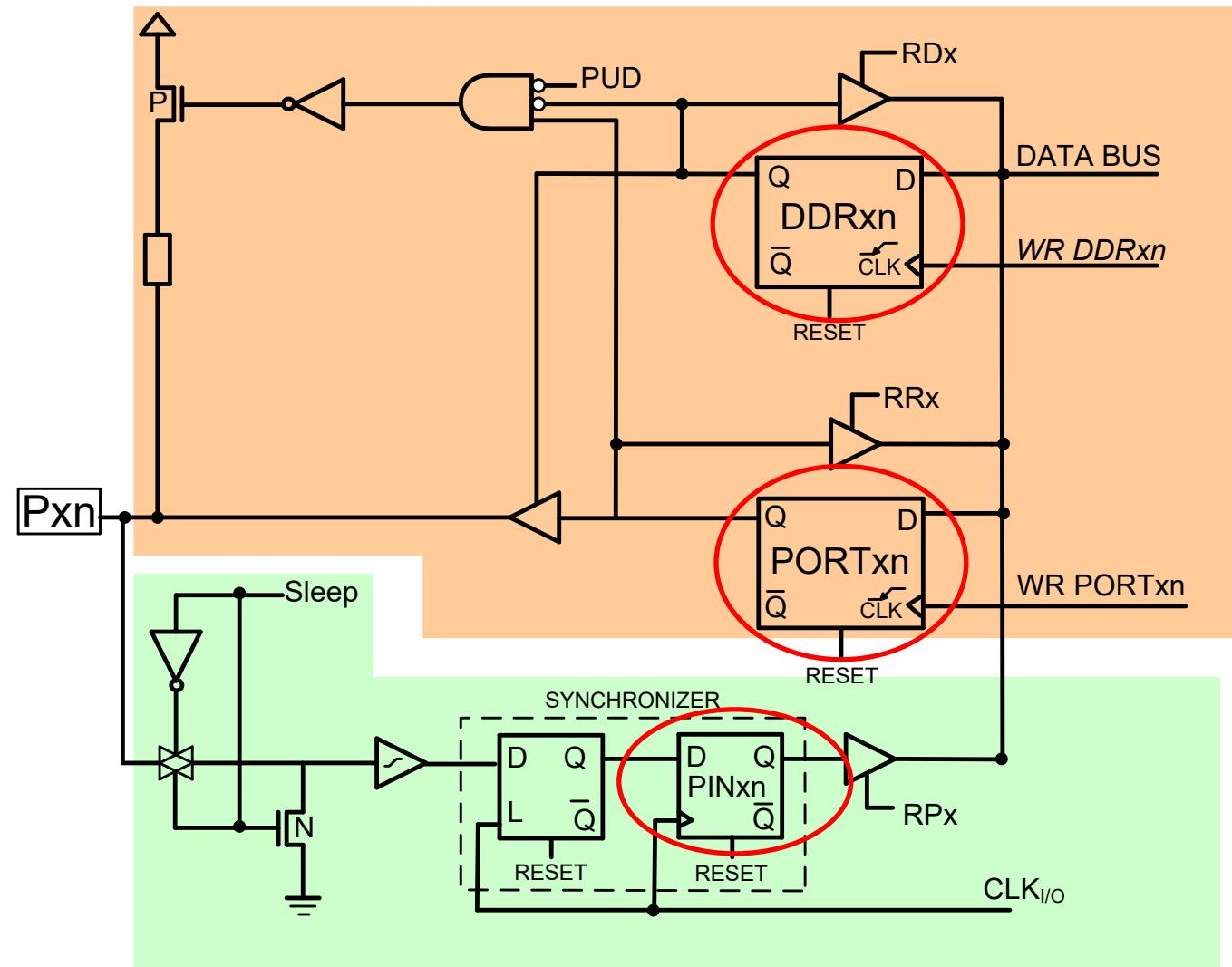
PORT-registrene

- Når et ben er sat op som **UDGANG**, kan man sende **0 volt** ud på benet ved at skrive et **0** på den tilsvarende bitplads i PORT-registeret.
- Når et ben er sat op som **UDGANG**, kan man sende **5 volt** ud på benet ved at skrive et **1** på den tilsvarende bitplads i PORT-registeret.

PIN-registrene

- Man kan **altid** aflæse, om der er **0 volt** eller **5 volt** på de enkelte ben på en port ved at **aflæse PIN-registeret**. Hvert ben modsvares af et bit i PIN-registeret.
- Hvis et bit i PIN-registeret aflæses til **0**, er der **0 volt** på det tilsvarende ben.
- Hvis et bit i PIN-registeret aflæses til **1**, er der **5 volt** på det tilsvarende ben.

(The structure of I/O pins)



Test ("socrative.com"): Room = AUQUIZ)

- Vi ønsker, at alle ben på PD skal være udgange.
Hvordan gøres det ?
- A: CLR R16
OUT DDRD,R16
- B: CLR R16
OUT PORTD,R16
- C: SER R16
OUT DDRD,R16
- D: SER R16
OUT PIND,R16

(I/O adresser for porte i Mega2560)

"IN eller OUT"

0x0B (0x2B)	PORTD
0x0A (0x2A)	DDRD
0x09 (0x29)	PIND
0x08 (0x28)	PORTC
0x07 (0x27)	DDRC
0x06 (0x26)	PINC
0x05 (0x25)	PORTB
0x04 (0x24)	DDRB
0x03 (0x23)	PINB
0x02 (0x22)	PORTA
0x01 (0x21)	DDRA
0x00 (0x20)	PINA

"IN eller OUT"

0x14 (0x34)	PORTG
0x13 (0x33)	DDRG
0x12 (0x32)	PING
0x11 (0x31)	PORTF
0x10 (0x30)	DDRF
0x0F (0x2F)	PINF
0x0E (0x2E)	PORTE
0x0D (0x2D)	DDRE
0x0C (0x2C)	PINE

(I/O adresser for porte i Mega2560)

"LDS eller STS"

(0x10B)	PORTL
(0x10A)	DDRL
(0x109)	PINL
(0x108)	PORTK
(0x107)	DDRK
(0x106)	PINK
(0x105)	PORTJ
(0x104)	DDRJ
(0x103)	PINJ
(0x102)	PORTH
(0x101)	DDRH
(0x100)	PINH

OBS:

For portene
H, J, K og L
Kan man **IKKE**
anvende **IN- eller**
OUT instruktioner

Example 1

- Write a program that makes all the pins of PORTA one.

```
LDI R20,0xFF ;R20 = 11111111 (binary)
OUT DDRA,R20 ;DDRA = R20
OUT PORTA,R20 ;PORTA = R20
```

DDRA: 1 1 1 1 1 1 1 1

PORTA: 1 1 1 1 1 1 1 1

PORTx	DDR _x	0	1
0		high impedance	Out 0
1		pull-up	Out 1

Example 2

- The following code will toggle all 8 bits of Port B forever with some time delay between “on” and “off” states:

```
LDI R16,0xFF          ;R16 = 0xFF = 0b11111111
OUT DDRB,R16          ;make Port B an output port (1111 1111)
L1:   LDI R16,0b01010101
      OUT PORTB,R16    ;put 01010101 on port B pins
      CALL DELAY
      LDI R16,0b10101010
      OUT PORTB,R16    ;put 10101010 on port B pins
      CALL DELAY
      JMP L1
```

Example 3

- A 7-segment is connected to PORTC. Display “1” on the 7-segment.

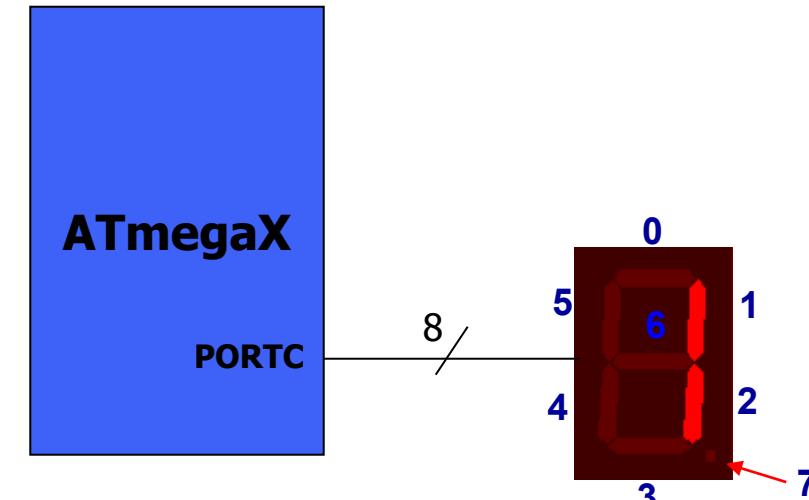
DDRC:

1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---

PORTC:

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

```
LDI R20, 0b11111111  
OUT DDRC, R20  
LDI R20, 0b00000110  
OUT PORTC, R20  
L1: JMP L1
```



PORTx	DDR _x	0	1
0	high impedance	Out 0	
1	pull-up		Out 1

Example 4

- A 7-segment is connected to PORTC. Display “3” on the 7-segment.

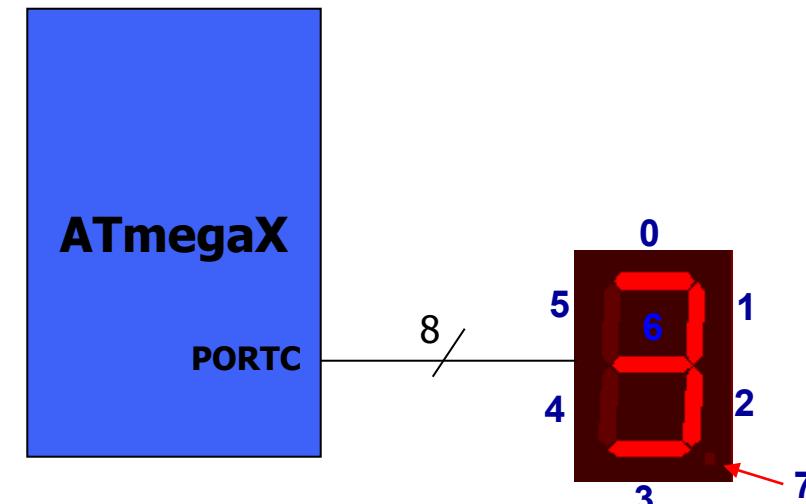
DDRC:

1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---

PORTC:

0	1	0	0	1	1	1	1
---	---	---	---	---	---	---	---

```
LDI R20, 0b11111111  
OUT DDRC, R20  
  
LDI R20, 0b01001111  
OUT PORTC, R20  
  
L1: JMP L1
```



PORTx	DDRx	0	1
0	high impedance	Out 0	
1	pull-up		Out 1

Input eksempel

- Skriv et program, som læser fra PA benene og skriver det til PB benene.

```
LDI R20,0
OUT DDRA,R20
LDI R20,0b11111111
OUT DDRB,R20
L1: IN R20,PINA
OUT PORTB,R20
JMP L1
```

PORTx	DDRx	0	1
0	high impedance	Out 0	
1	pull-up		Out 1

Single-Bit I/O instruktioner

Table 4-7: Single-Bit (Bit-Oriented) Instructions for AVR

Instruction	Function
SBI ioReg,bit	Set Bit in I/O register (set the bit: bit = 1)
CBI ioReg,bit	Clear Bit in I/O register (clear the bit: bit = 0)
SBIC ioReg,bit	Skip if Bit in I/O register Cleared (skip next instruction if bit = 0)
SBIS ioReg,bit	Skip if Bit in I/O register Set (skip next instruction if bit = 1)

OBS: Virker kun for I/O registre med
adresser lavere end 0x20

SBI og CBI instruktioner

- **SBI** (Set Bit in IO register)
 - SBI ioReg, bit
 - Eksempler:
 - **SBI PORTD, 0** ;PORTD bit 0 = 1
 - **SBI DDRC, 5** ;DDRC bit 5 = 1
- **CBI** (Clear Bit in IO register)
 - CBI ioReg, bit
 - Eksempler:
 - **CBI PORTD, 0** ;PORTD bit 0 = 0
 - **CBI DDRC, 5** ;DDRC bit 5 = 0

Eksempel

- Skriv et program, som kontinuert toggler PA ben 4.

```
SBI DDRA,4      ;PA ben 4 er udgang  
L1: SBI PORTA,4    ;PA ben 4 = 1 (5 volt)  
      CBI PORTA,4    ;PA ben 4 = 0 (0 volt)  
      JMP L1         ;Gentag altid
```

SBIC og SBIS

- **SBIC** (Skip if Bit in IO register Cleared)

- SBIC ioReg, bit ; if (ioReg.bit = 0) skip next instruction
- Eksempel:

```
SBIC PIND, 0      ;skip next instruction if PIND bit 0 = 0
INC   R20          ;this instruction MIGHT be skipped
LDI   R19, 0x23    ;this instruction always executes
```

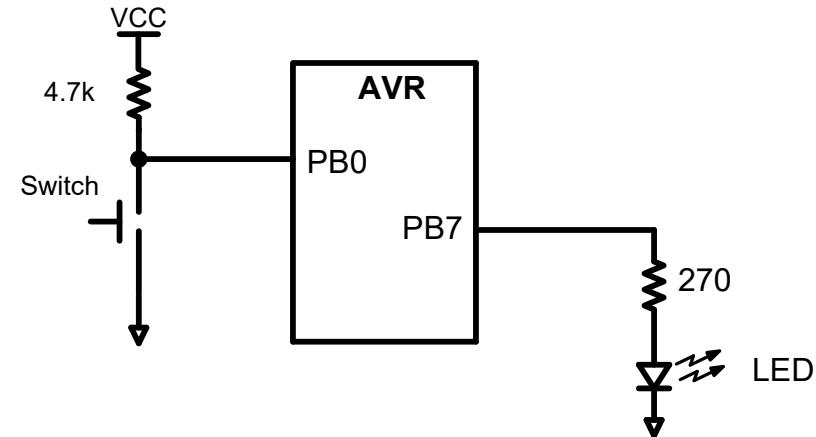
- **SBIS** (Skip if Bit in IO register Set)

- SBIS ioReg, bit ; if (ioReg.bit = 1) skip next instruction
- Eksempel:

```
SBIS PIND, 0      ;skip next instruction if PIND bit 0 = 1
INC   R20          ;this instruction MIGHT be skipped
LDI   R19, 0x23    ;this instruction always executes
```

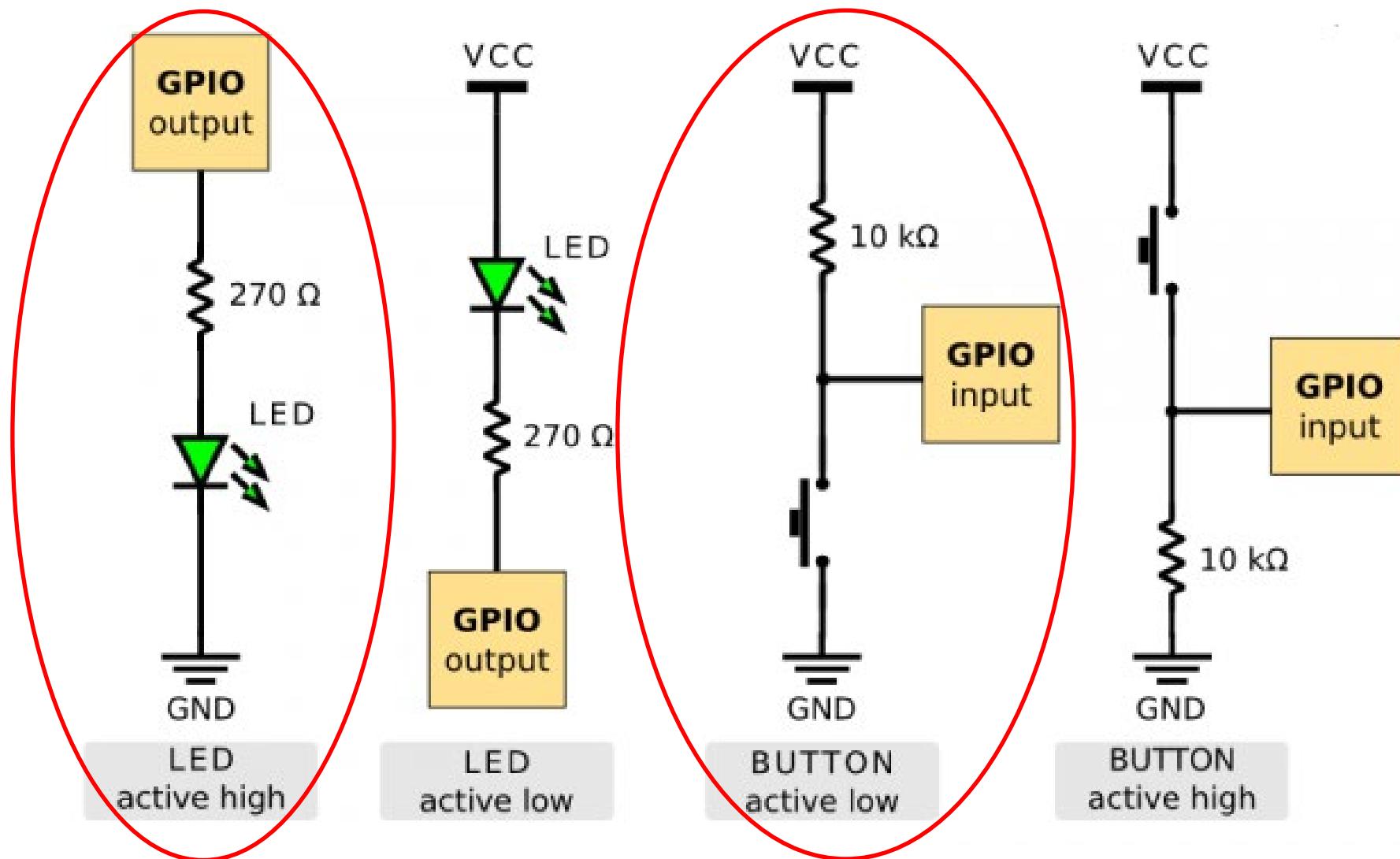
Eksempel

- En switch er forbundet til PB ben 0 og en lysdiode til PB ben 7. Skriv et program, der slukker lysdioden, hvis der trykkes på knappen. Ellers skal den lyse.



```
CBI DDRB,0           ;make PB0 an input
SBI DDRB,7           ;make PB7 an output
AGAIN: SBIC PINB,0   ;skip next if switch PRESSED
        JMP OVER
        CBI PORTB,7    ;turn off LED
        JMP AGAIN
OVER:  SBI PORTB,7   ;turn on LED
        JMP AGAIN
```

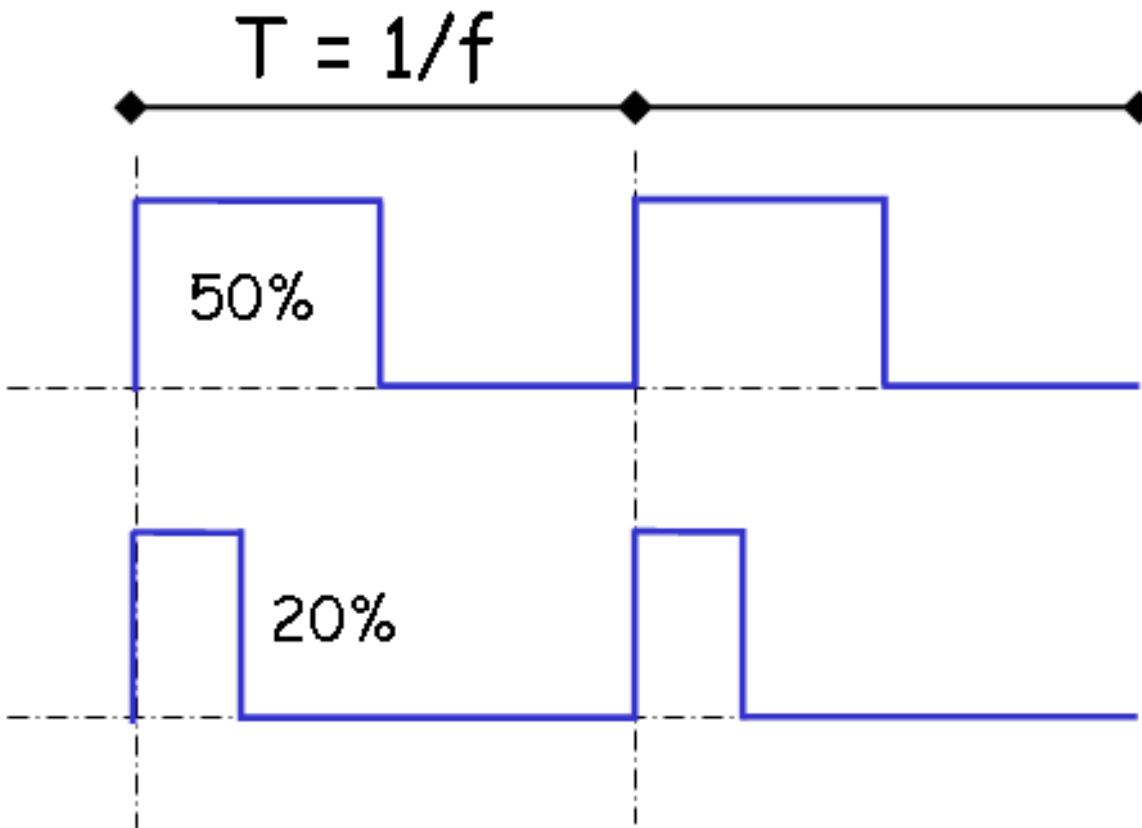
Active High / Active Low



Shield

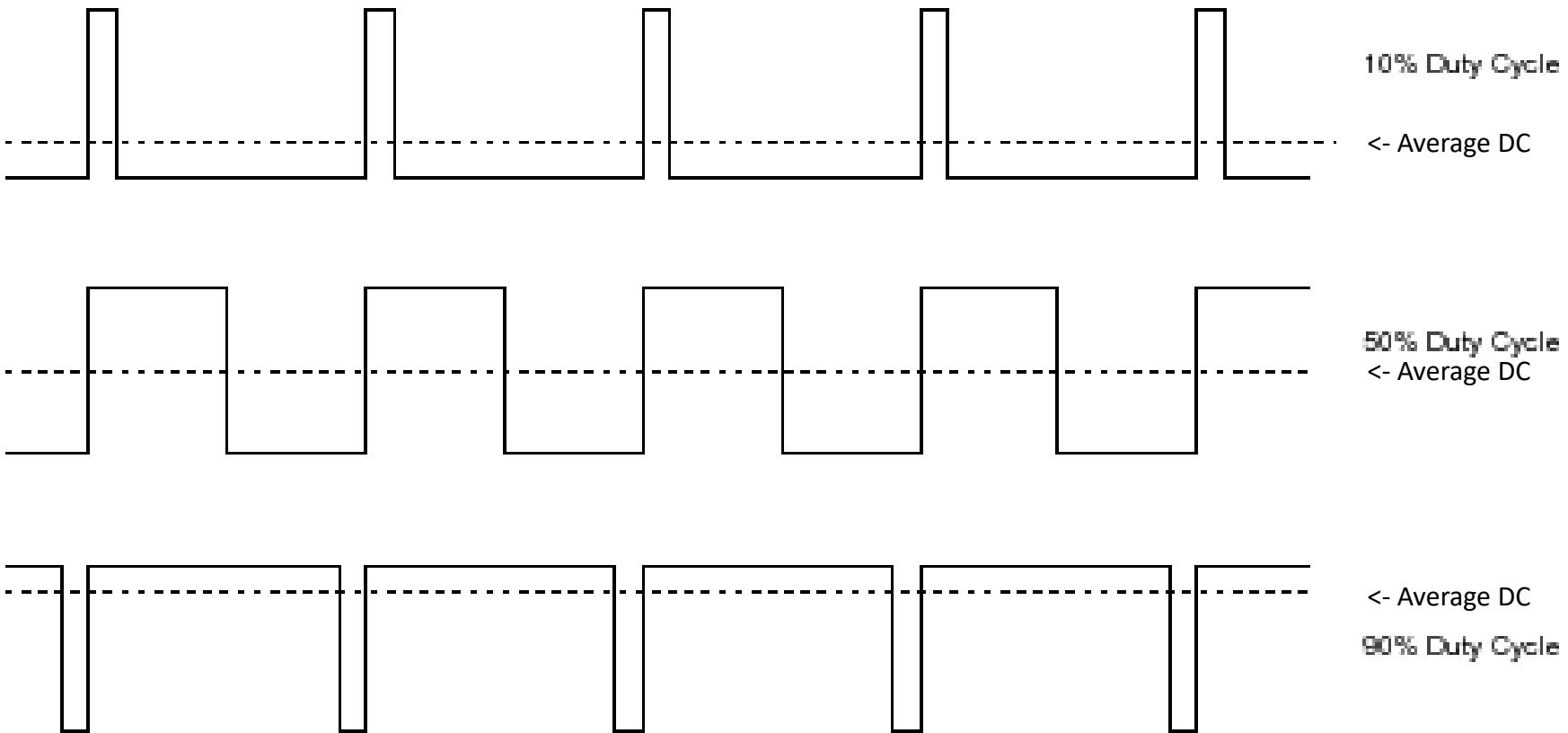
- SW0-SW7 er forbundet til PA0-PA7
- LED0-LED7 er forbundet til PB0-PB7

PWM = Pulse Width Modulation

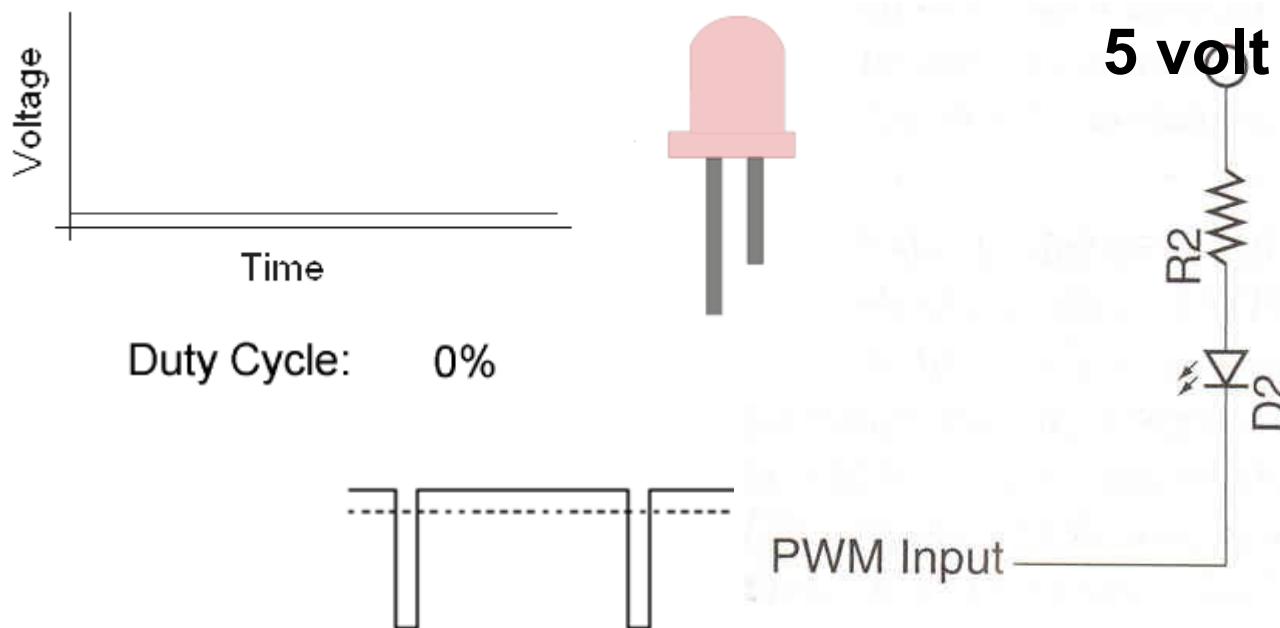
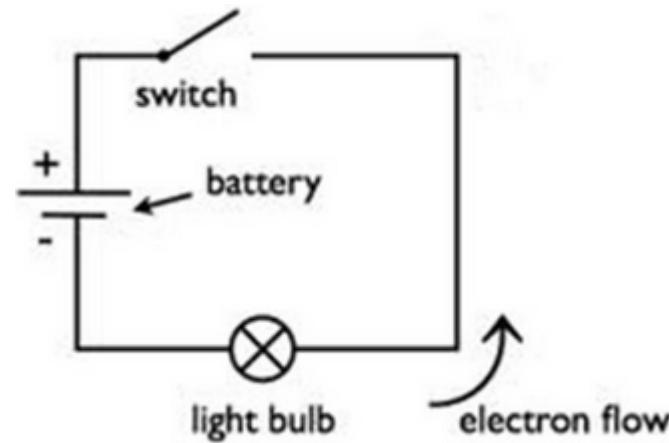


**PWM: Vi ønsker af styre "duty cycle"
(frekvensen er mindre vigtig)**

Forskellige "duty cycles"



PWM : Styring af lysintensitet



Test ("socrative.com"): Room = AUQUIZ)

- Hvilken duty cycle får vi for signalet på PB bit 6 ?

SBI DDRB, 6

IGEN:

```
SBI PORTB, 6
CALL DELAY
CBI PORTB, 6
CALL DELAY
CALL DELAY
JMP IGEN
```

- A: 33 %
- B: 25 %
- C: 50 %
- D: 66 %

CPI - (Compare with Immediate)

Description:

This instruction performs a compare between register Rd and a constant. The register is not changed. All conditional branches can be used after this instruction.

Operation:

- (i) $Rd - K$

Syntax:

(i) CPI Rd,K

Operands:

$16 \leq d \leq 31, 0 \leq K \leq 255$

Program Counter:

$PC \leftarrow PC + 1$

16-bit Opcode:

0011	KKKK	dddd	KKKK
------	------	------	------

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
-	-	\Leftrightarrow	\Leftrightarrow	\Leftrightarrow	\Leftrightarrow	\Leftrightarrow	\Leftrightarrow

Example:

```
cpi      r19,3      ; Compare r19 with 3
brne    error        ; Branch if r19<>3
...
error:   nop         ; Branch destination (do nothing)
```

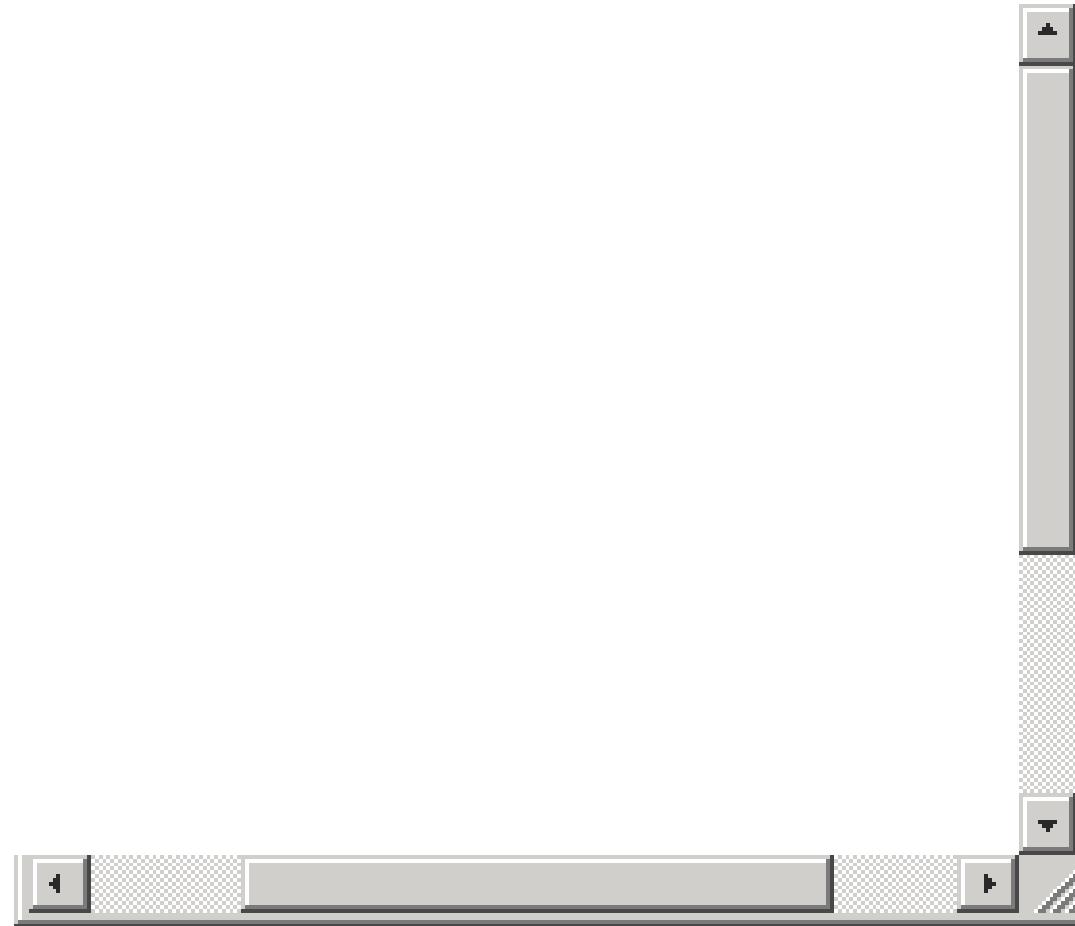
LAB4

Der skal nu laves ændringer i programmet, således at programmet ikke længere "automatisk" spiller de 8 toner, men vi ønsker i stedet at kunne anvende de 8 trykknapper ("SW7 – SW0") på "Mega2560 I/O Shield" som "tangenter" på et klaver:



SW. : 7 6 5 4 3 2 1 0

Slut



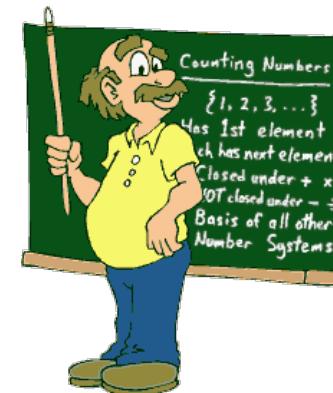


MSYS

Microcontroller Systems

Lektion 9

Aritmetiske og logiske instruktioner



Slides from Henning Hargaard

Binær addition

$$\begin{array}{r} X \quad 190 \\ Y \quad + 141 \\ \hline X + Y \quad 331 \end{array}$$

A binary addition diagram showing the sum of 190 and 141. The result 331 is circled in red. Blue arrows point from the decimal digits 190 to the corresponding binary digits 1011110.

$$\begin{array}{r} X \quad 173 \\ Y \quad + 44 \\ \hline X + Y \quad 217 \end{array}$$

A binary addition diagram showing the sum of 173 and 44. The result 217 is circled in red. Blue arrows point from the decimal digits 173 to the corresponding binary digits 1010110.

Figure 2–1 Examples of decimal and corresponding binary additions.

11
10
01
00

Binær subtraktion

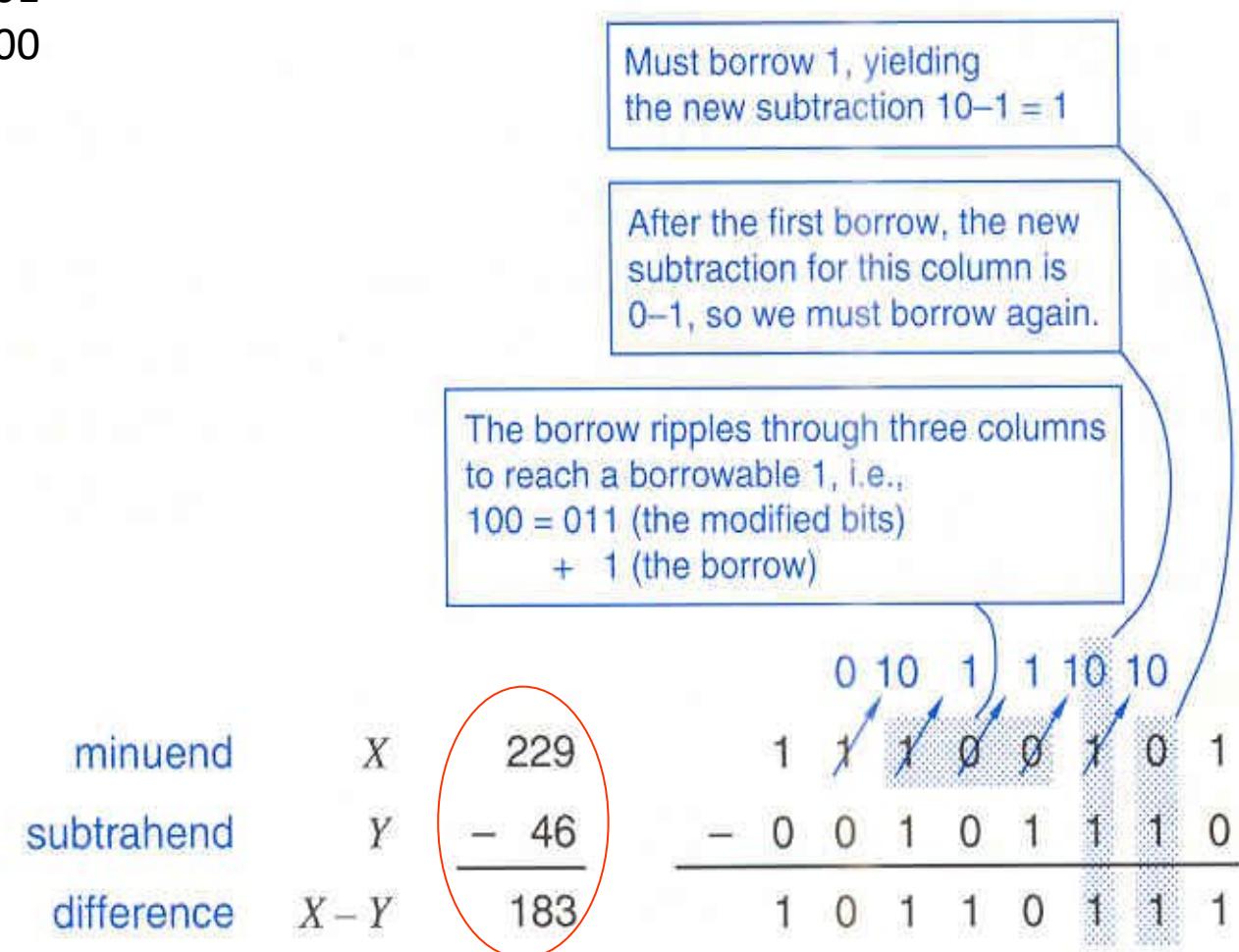
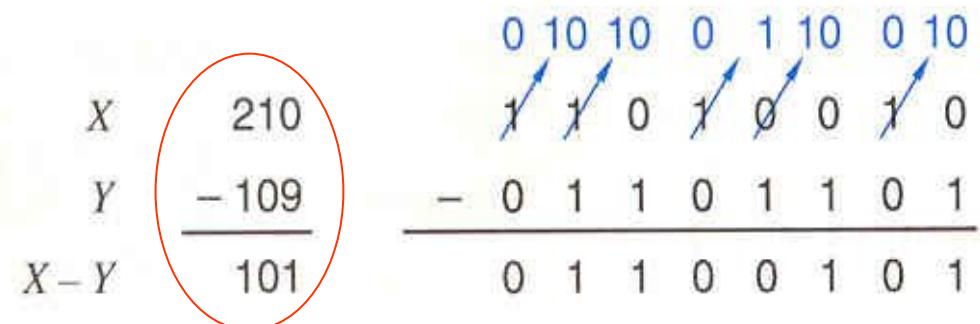
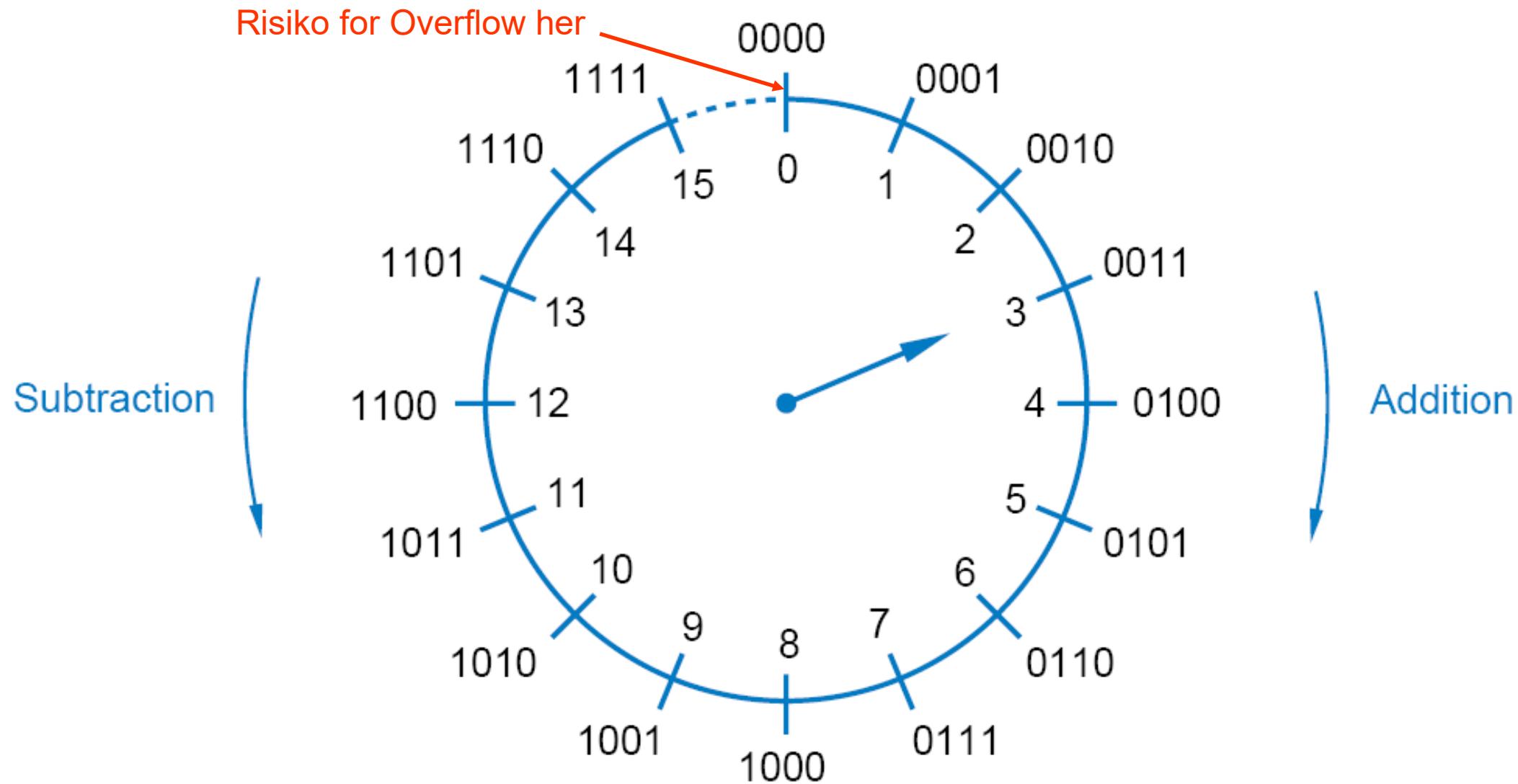


Figure 2–2
Examples of decimal and corresponding binary subtractions.



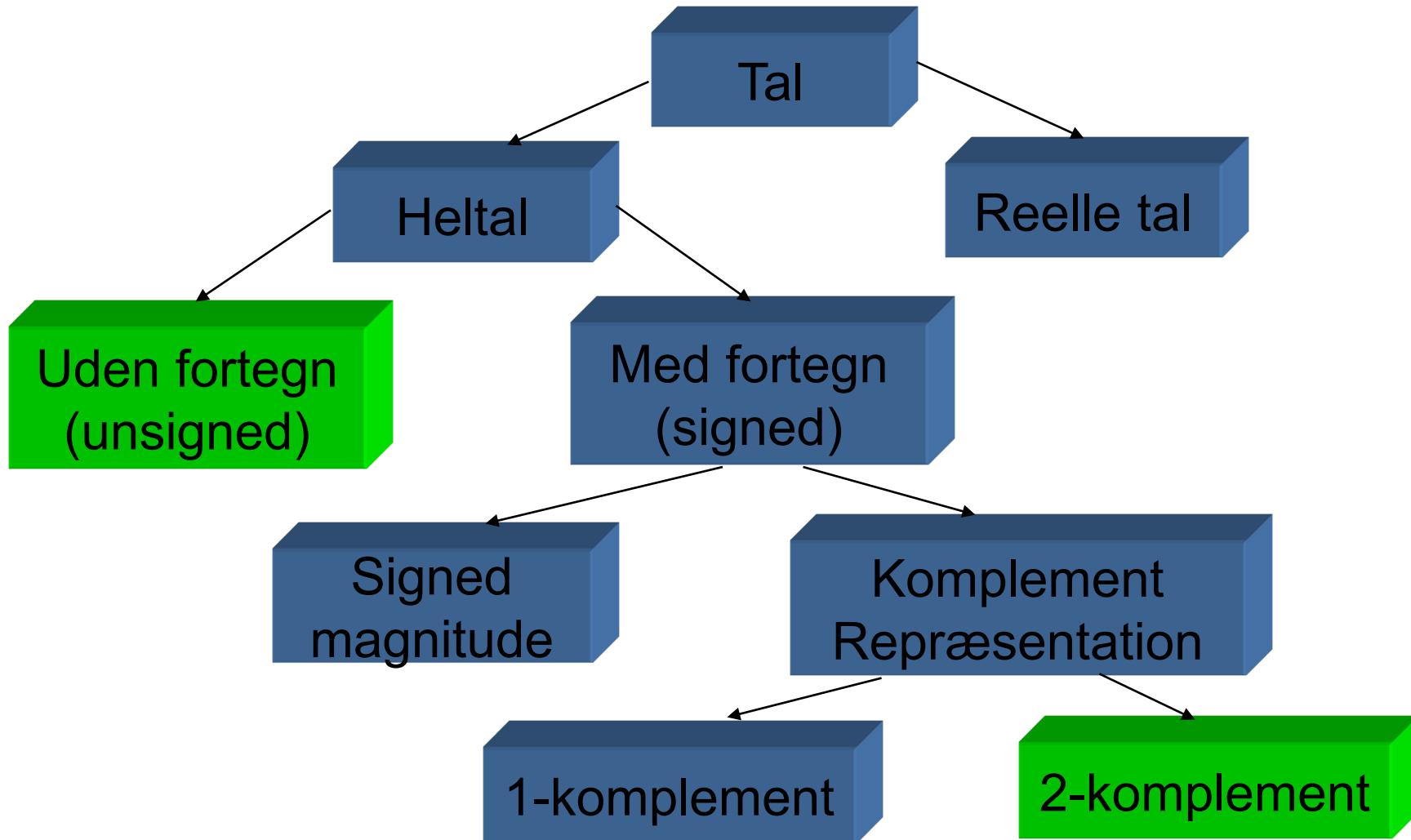
Unsigned addition / subtraktion



Hvad står der her ?

10001000

Tal-repræsentationer



Signed magnitude

- Vi anvender typisk MSB til at repræsentere fortegnet.

$$01010101_2 = +85_{10}$$

$$01111111_2 = +127_{10}$$

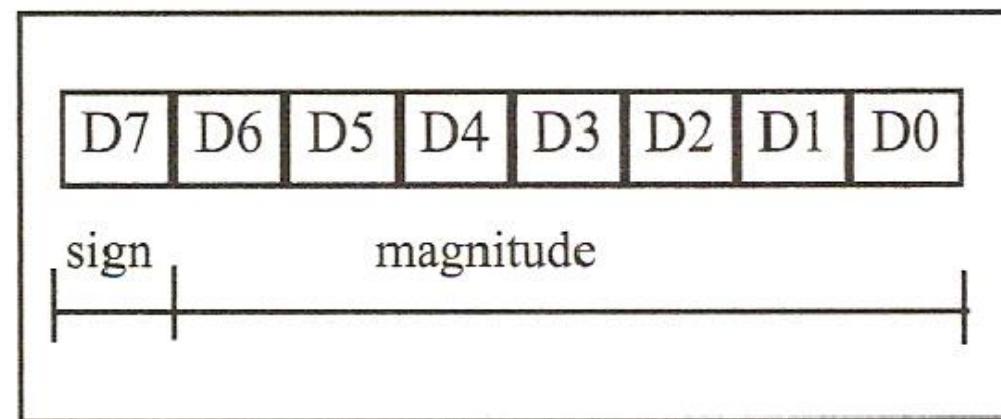
$$\text{00000000}_2 = +0_{10}$$

$$11010101_2 = -85_{10}$$

$$11111111_2 = -127_{10}$$

$$\text{10000000}_2 = -0_{10}$$

- Bemærk, at vi har 2 værdier, der begge repræsenterer 0 !



Dannelsen af 1-komplement

A **ones' complement system** or **ones' complement arithmetic** is a system in which negative numbers are represented by the inverse of the binary representations of their corresponding positive numbers

3 bit ones'-complement:

Value	Bits
3	011
2	010
1	001
0	000
-0	111
-1	110
-2	101
-3	100

$$17_{10} = 00010001_2$$



$$11101110_2 = -17_{10}$$

$$-99_{10} = 10011100_2$$



$$01100011_2 = 99_{10}$$

$$119_{10} = 01110111_2$$



$$10001000_2 = -119_{10}$$

$$-127_{10} = 10000000_2$$



$$01111111_2 = 127_{10}$$

$$0_{10} = 00000000_2 \text{ (positive zero)}$$



$$11111111_2 = 0_{10} \text{ (negative zero)}$$

- Bemærk : 2 værdier for 0 !

Dannelsen af 2-komplement

Two's complement is the most common method of representing signed (positive, negative, and zero) integers on computers.

Regel : Inverter alle bit og læg en til.

3 bit two's-complement:

Value	Bits
3	011
2	010
1	001
0	000
-1	111
-2	110
-3	101
-4	100

$$17_{10} = 00010001_2$$

↓ complement bits

$$\begin{array}{r} 11101110 \\ +1 \\ \hline 11101111_2 \end{array} = -17_{10}$$

$$-99_{10} = 10011101_2$$

↓ complement bits

$$\begin{array}{r} 01100010 \\ +1 \\ \hline 01100011_2 \end{array} = 99_{10}$$

$$119_{10} = 01110111_2$$

↓ complement bits

$$\begin{array}{r} 10001000 \\ +1 \\ \hline 10001001_2 \end{array} = -119_{10}$$

$$-127_{10} = 10000001_2$$

↓ complement bits

$$\begin{array}{r} 01111110 \\ +1 \\ \hline 01111111_2 \end{array} = 127_{10}$$

$$0_{10} = 00000000_2$$

↓ complement bits

$$\begin{array}{r} 11111111 \\ +1 \\ \hline 10000000_2 \end{array} = 0_{10}$$

$$-128_{10} = 10000000_2$$

↓ complement bits

$$\begin{array}{r} 01111111 \\ +1 \\ \hline 10000000_2 \end{array} = -128_{10}$$

Nummer systemer

Table 2-6 Decimal and 4-bit numbers.

<i>Decimal</i>	<i>Two's Complement</i>	<i>Ones' Complement</i>	<i>Signed Magnitude</i>
-8	1000	—	—
-7	1001	1000	1111
-6	1010	1001	1110
-5	1011	1010	1101
-4	1100	1011	1100
-3	1101	1100	1011
-2	1110	1101	1010
-1	1111	1110	1001
0	0000	1111 or 0000	1000 or 0000
1	0001	0001	0001
2	0010	0010	0010
3	0011	0011	0011
4	0100	0100	0100
5	0101	0101	0101
6	0110	0110	0110
7	0111	0111	0111

2-komplement addition

$$\begin{array}{r} +3 \\ +4 \\ \hline +7 \end{array} \quad \begin{array}{r} 0011 \\ + 0100 \\ \hline 0111 \end{array}$$

$$\begin{array}{r} -2 \\ + -6 \\ \hline -8 \end{array} \quad \begin{array}{r} 1110 \\ + 1010 \\ \hline 1\ 1000 \end{array}$$

$$\begin{array}{r} +6 \\ + -3 \\ \hline +3 \end{array} \quad \begin{array}{r} 0110 \\ + 1101 \\ \hline 1\ 0011 \end{array}$$

$$\begin{array}{r} +4 \\ + -7 \\ \hline -3 \end{array} \quad \begin{array}{r} 0100 \\ + 1001 \\ \hline 1101 \end{array}$$

- Bemærk : Samme hardware / instruktioner som ved unsigned behandling.
Det er blot **vores måde at kode tal på**, der er anderledes !

"2-komplement – uret"



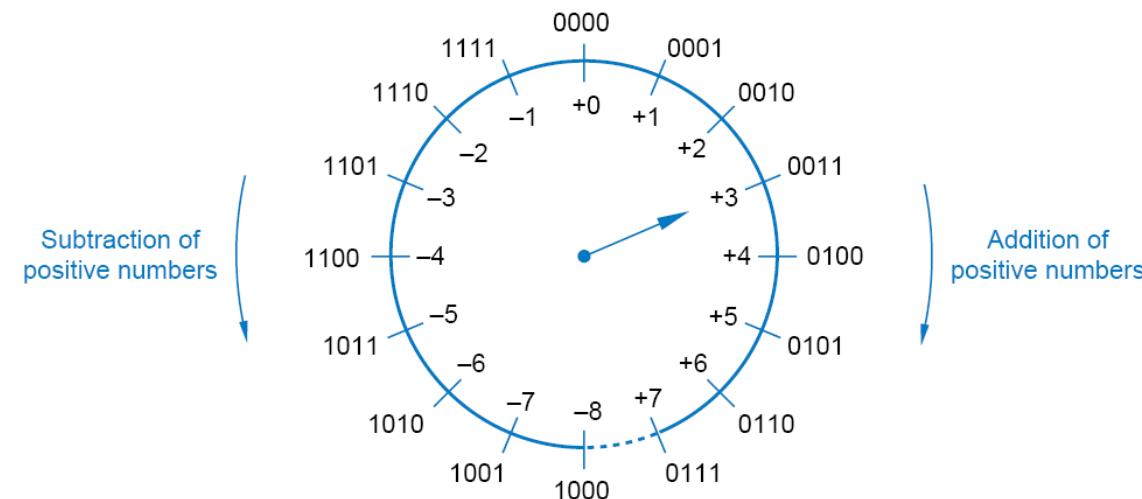
2-komplement Overflow

$$\begin{array}{r} -3 \\ + -6 \\ \hline -9 \end{array} \quad \begin{array}{r} 1101 \\ + 1010 \\ \hline 10111 \end{array} = +7$$

$$\begin{array}{r} +5 \\ + +6 \\ \hline +11 \end{array} \quad \begin{array}{r} 0101 \\ + 0110 \\ \hline 1011 \end{array} = -5$$

$$\begin{array}{r} -8 \\ + -8 \\ \hline -16 \end{array} \quad \begin{array}{r} 1000 \\ + 1000 \\ \hline 10000 \end{array} = +0$$

$$\begin{array}{r} +7 \\ + +7 \\ \hline +14 \end{array} \quad \begin{array}{r} 0111 \\ + 0111 \\ \hline 1110 \end{array} = -2$$



Test ("socrative.com"): Room = AUQUIZ

Hvordan gemmes tallet -5 (= minus 5) i
"signed magnitude" formatet ?

A: 0b00000101

B: 0b10000101

C: 0b11111010

D: 0b11111011

Test ("socrative.com"): Room = AUQUIZ

Hvordan gemmes tallet -5 (= minus 5) i
"1's complement" formatet ?

A: 0b00000101

B: 0b10000101

C: 0b11111010

D: 0b11111011

Test ("socrative.com"): Room = AUQUIZ

Hvordan gemmes tallet -5 (= minus 5) i
"2's complement" formatet ?

A: 0b00000101

B: 0b10000101

C: 0b11111010

D: 0b11111011

Additions Instruktioner

ADD Rd,Rr ;Rd = Rd + Rr

ADC Rd,Rr ;Rd = Rd + Rr + C

(ADIW Rd:Rd,K ;Rd+1:Rd = Rd+1:Rd + K)

ADD (Add without Carry)

Description:

Adds two registers **without the C Flag** and places the result in the destination register Rd.

Operation:

(i) $Rd \leftarrow Rd + Rr$

Syntax:

(i) ADD Rd,Rr

Operands:

$0 \leq d \leq 31, 0 \leq r \leq 31$

Program Counter:

$PC \leftarrow PC + 1$

16-bit Opcode:

0000	11rd	dddd	rrrr
------	------	------	------

Example:

add r1,r2 ; Add r2 to r1 ($r1=r1+r2$)

add r28,r28 ; Add r28 to itself ($r28=r28+r28$)

ADC (Add with Carry)

Description:

Adds two registers and the contents of the C Flag and places the result in the destination register Rd.

Operation:

$$(i) \quad Rd \leftarrow Rd + Rr + C$$

Syntax:

(i) ADC Rd,Rr

Operands:

$0 \leq d \leq 31, 0 \leq r \leq 31$

Program Counter:

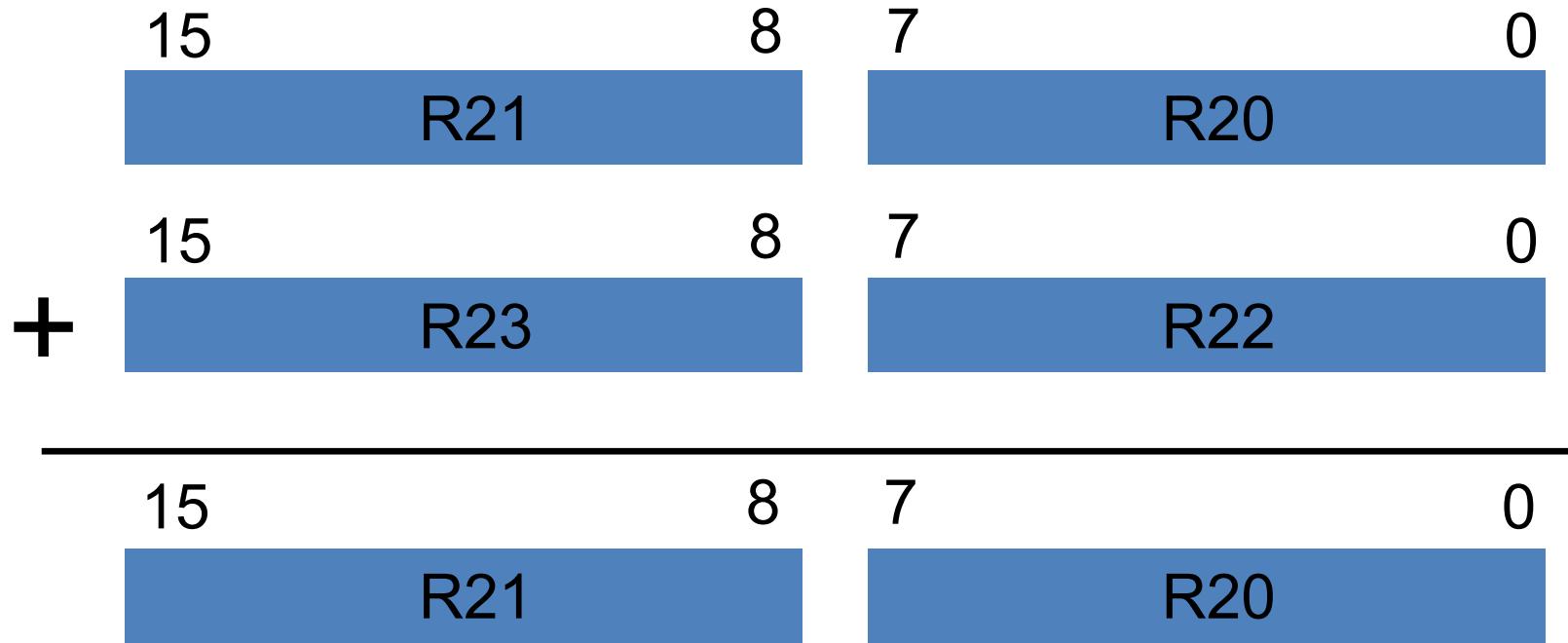
$PC \leftarrow PC + 1$

Example:

```
; Add R1:R0 to R3:R2  
add  r2,r0      ; Add low byte  
adc  r3,r1      ; Add with carry high byte
```

Addition af 16 bit tal

OPGAVE: Skriv kode, der lægger to 16 bit tal sammen.



Hint: Gør brug af instruktionerne ADD og ADC.

Subtraktions Instruktioner

SUB Rd,Rr

;Rd = Rd – Rr

SBC Rd,Rr

;Rd = Rd - Rr - C

SUBI Rd,K

;Rd = Rd - K

SBCI Rd,K

;Rd = Rd - K – C

SBIW Rd:Rd,K

;Rd+1:Rd = Rd+1:Rd - K

Vores microcontroller anvender denne metode for subtraktion
(den "lægger sammen for at trække fra"):

1. Danner 2-komplementet af tallet, der skal subtraheres.
2. Adderer de to tal.
3. Inverterer Carry-flaget.

Multiplication

Table 5-1: Multiplication Summary

Multiplication	Application	Byte1	Byte2	High byte of result	Low byte of result
MUL Rd, Rr	Unsigned numbers	Rd	Rr	R1	R0
MULS Rd, Rr	Signed numbers	Rd	Rr	R1	R0
MULSU Rd, Rr	Unsigned numbers with signed numbers	Rd	Rr	R1	R0

The following example multiplies 25H by 65H.

```
LDI    R23,0x25      ;load 25H to R23
LDI    R24,0x65      ;load 65H to R24
MUL    R23,R24       ;25H * 65H = E99 where
                      ;R1 = 0EH and R0 = 99H
```

Eksempel på divisions-algoritme (s.167)

```
.DEF NUM = R20
.DEF DENOMINATOR = R21
.DEF QUOTIENT = R22

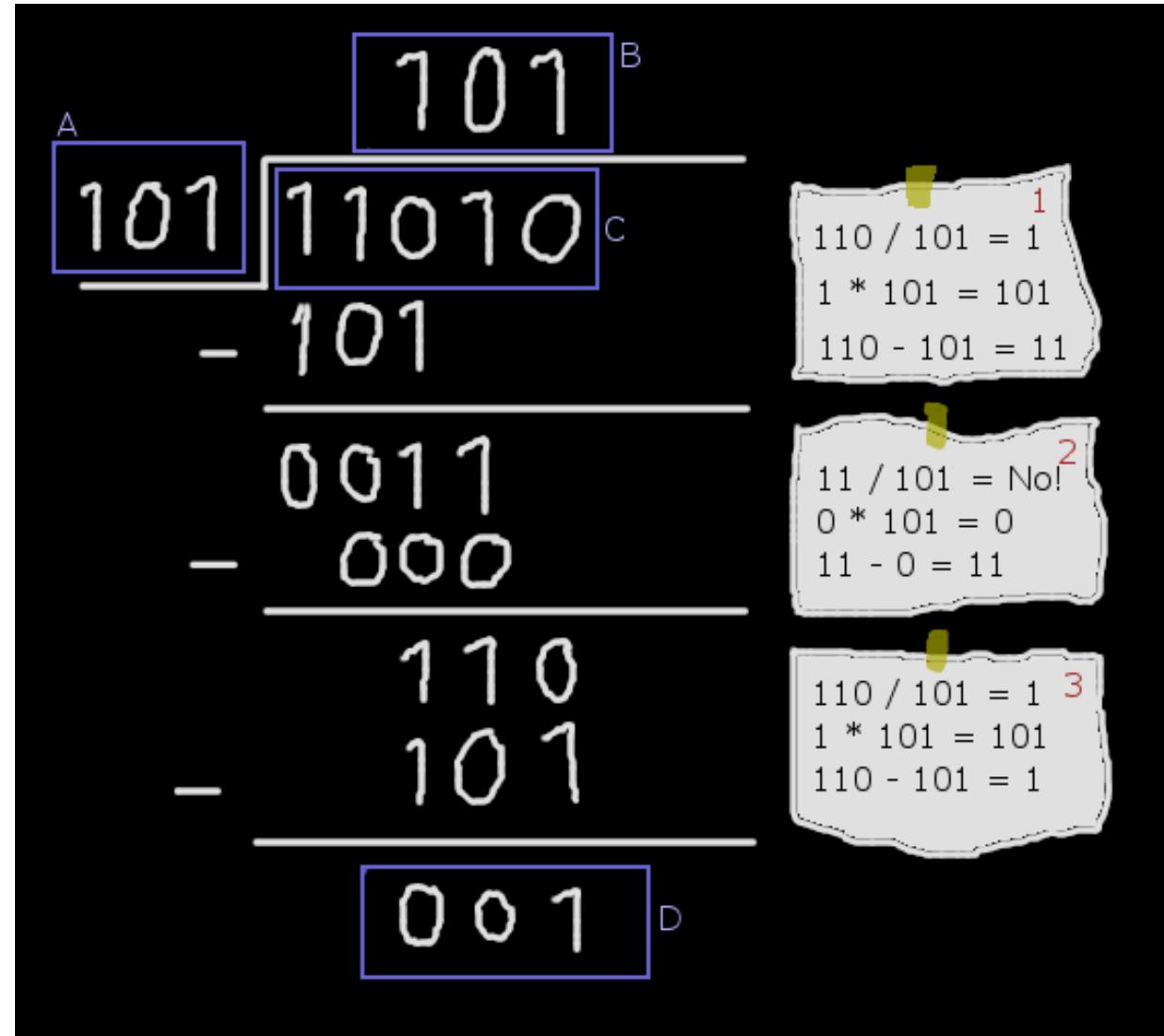
LDI NUM, 95 ;NUM = 95
LDI DENOMINATOR, 10 ;DENOMINATOR = 10
CLR QUOTIENT ;QUOTIENT = 0

L1: INC QUOTIENT
    SUB NUM, DENOMINATOR
    BRCC L1 ;branch if C is zero

    DEC QUOTIENT ;once too many
    ADD NUM, DENOMINATOR ;add back to it

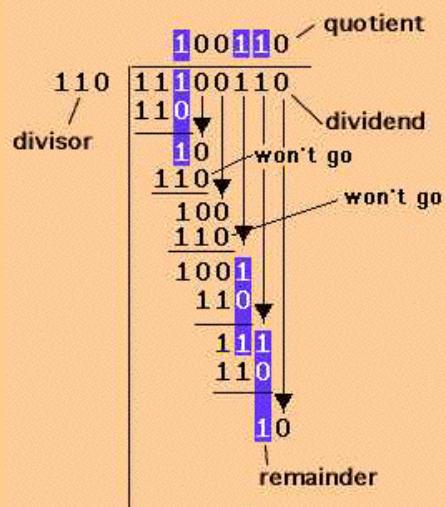
HERE: JMP HERE ;stay here forever
```

Binær division med shift / subtract



Binær division med shift / subtract

- Set quotient to 0
- Align leftmost digits in dividend and divisor
- **Repeat**
 - If that portion of the dividend above the divisor is greater than or equal to the divisor
 - Then subtract divisor from that portion of the dividend and
 - Concatenate 1 to the right hand end of the quotient
 - Else concatenate 0 to the right hand end of the quotient
 - Shift the divisor one place right
- Until dividend is less than the divisor
- quotient is correct, dividend is remainder
- STOP



AND Rd,Rr (Rd = Rd AND Rr)

Logical AND Function

<u>Inputs</u>	<u>Output</u>	
X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

X ———|———| X AND Y
Y ———|———|

Vælegnet til at **NULSTILLE** bestemte bit.

Også muligt at AND'e med en konstant:

ANDI Rd,K ;rd = Rd AND K

AND eksempel

Example 5-18

Show the results of the following.

```
LDI    R20,0x35      ;R20 = 35H  
ANDI   R20,0x0F      ;R20 = R20 AND 0FH (now R20 = 05)
```

Solution:

	35H	0011 0101	
AND	0FH	0000 1111	

	05H	0000 0101	; 35H AND 0FH = 05H, Z = 0, N = 0

OR Rd,Rr ($Rd = Rd \text{ OR } Rr$)

Logical OR Function

Inputs		Output
X	Y	$X \text{ OR } Y$
0	0	0
0	1	1
1	0	1
1	1	1

X
Y

Velegnet til at **SÆTTE** bestemte bit.

Også muligt at OR'e med en konstant:

ORI Rd,K ;rd = Rd OR K

OR eksempel

Example 5-19

(a) Show the results of the following:

LDI	R20, 0x04	; R20 = 04
ORI	R20, 0x30	; now R20 = 34H

(b) Assume that PB2 is used to control an outdoor light, and PB5 to control a light inside a building. Show how to turn “on” the outdoor light and turn “off” the inside one.

Solution:

(a)

04H	0000 0100	
OR	30H	0011 0000

34H 0011 0100 04 OR 30 = 34H, Z = 0 and N = 0

(b)

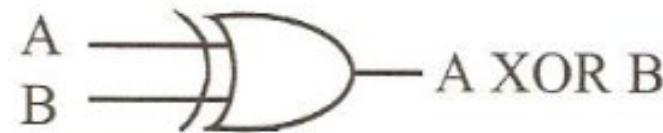
SBI	DDRB, 2	; bit 2 of Port B is output
SBI	DDRB, 5	; bit 5 of Port B is output
IN	R20, PORTB	; move PORTB to R20. (Notice that we read ; the value of PORTB instead of PINB ; because we want to know the last value ; of PORTB, not the value of the AVR ; chip pins.)
ORI	R20, 0b00000100	; set bit 2 of R20 to one
ANDI	R20, 0b11011111	; clear bit 5 of R20 to zero
OUT	PORTB, R20	; out R20 to PORTB

HERE: JMP HERE ; stop here

EOR Rd,Rr ($Rd = Rd \text{ XOR } Rr$)

Logical XOR Function

<u>Inputs</u>		<u>Output</u>
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0



Velegnet til at **INVERTERE** bestemte bit.

EOR eksempel

Example 5-20

Show the results of the following:

LDI R20, 0x54

LDI R21, 0x78

EOR R20, R21

Solution:

	54H	0101	0100
XOR	78H	0111	1000

	2CH	0010	1100

54H XOR 78H = 2CH, Z = 0, N = 0

Test ("socrative.com"): Room = AUQUIZ)

Hvordan kan man nulstille bit 7 i R16 (uden at ændre på resten af bittene) ?

A: LDI R17, 0b10000000
AND R16, R17

B: LDI R17, 0b01111111
EOR R16, R17

C: LDI R17, 0b10000000
OR R16, R17

D: LDI R17, 0b01111111
AND R16, R17

Test ("socrative.com"): Room = AUQUIZ

Hvordan kan man sætte bit 7 i R16 (uden at ændre på resten af bittene) ?

- A: LDI R17, 0b10000000
AND R16, R17
- B: LDI R17, 0b01111111
EOR R16, R17
- C: LDI R17, 0b10000000
OR R16, R17
- D: LDI R17, 0b01111111
AND R16, R17

Test ("socrative.com"): Room = AUQUIZ

Hvordan kan man invertere / toggle bit 7 i R16
(uden at ændre på resten af bittene) ?

- A: LDI R17, 0b10000000
AND R16, R17
- B: LDI R17, 0b10000000
EOR R16, R17
- C: LDI R17, 0b10000000
OR R16, R17
- D: LDI R17, 0b01111111
AND R16, R17

Compare

CP Rd,Rr ;Rd sammenlignes med Rr

CPI Rd,K ;Rd sammenlignes med konstanten K

Table 5-2: AVR Compare Instructions

BREQ	Branch if equal	Branch if Z = 1
BRNE	Branch if not equal	Branch if Z = 0
BRSH	Branch if same or higher	Branch if C = 0
BRLO	Branch if lower	Branch if C = 1
BRLT	Branch if less than (signed)	Branch if S = 1
BRGE	Branch if greater than or equal (signed)	Branch if S = 0
BRVS	Branch if Overflow flag set	Branch if V = 1
BRVC	Branch if Overflow flag clear	Branch if V = 0

COM og NEG

COM (complement)

This instruction complements the contents of a register. The complement action changes the 0s to 1s, and the 1s to 0s. This is also called *1's complement*.

```
LDI R20, 0xAA ;R20 = 0xAA  
COM R20 ;now R20 = 55H
```

NEG (negate)

This instruction takes the 2's complement of a register. See Example 5-23.

Example 5-23

Find the 2's complement of the value 85H. Notice that 85H is -123.

Solution:

LDI R21, 0x85	;	85H = 1000 0101
		;1' s = 0111 1010
		+ 1
NEG R21	;	2' s comp 0111 1011 = 7BH

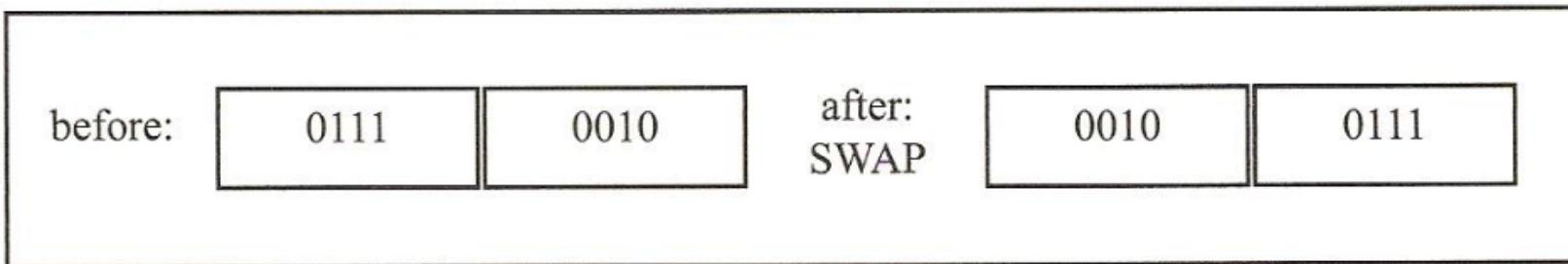
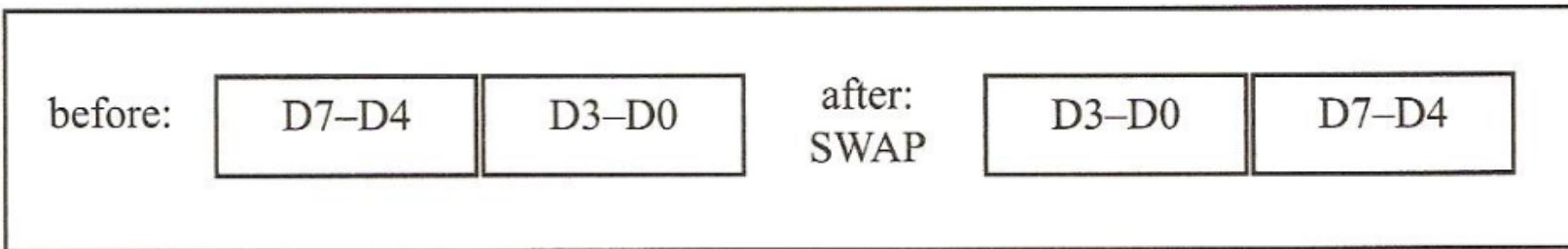
Logical Inverter

Input	Output
X	NOT X
0	1
1	0

X —————— NOT X

SWAP Register

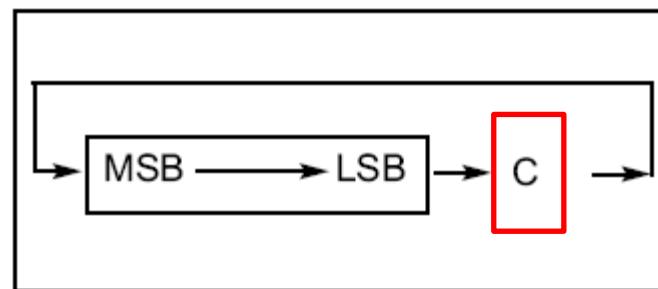
Eksempel: SWAP R16



ROR - Rotate Right through Carry

ROR Rd ;Rd (only flags are set)

In ROR, as bits are rotated from left to right, the carry flag enters the MSB and the LSB exits to the carry flag. In other words, **in ROR the C is moved to the MSB, and the LSB is moved to the C.**

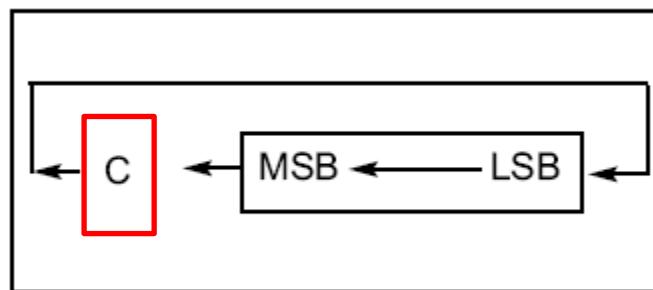


See what happens to 0010 0110 after running 3 ROR instructions:

CLC		;make C = 0 (carry is 0)
LDI	R20 , 0x26	;R20 = 0010 0110
ROR	R20	;R20 = 0001 0011 C = 0
ROR	R20	;R20 = 0000 1001 C = 1
ROR	R20	;R20 = 1000 0100 C = 1

ROL - Rotate Left trough Carry

In ROL, as bits are shifted from right to left, the carry flag enters the LSB and the MSB exits to the carry flag. In other words, **in ROL the C is moved to the LSB, and the MSB is moved to the C.**

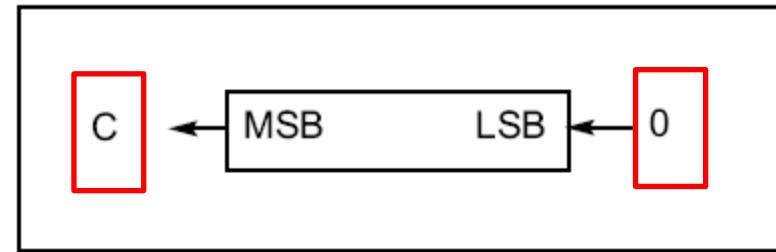


SEC	;make C = 1 (carry is 1)
LDI R20,0x15	;R20 = 0001 0101
ROL R20	;R20 = 0010 1011 C = 0
ROL R20	;R20 = 0101 0110 C = 0
ROL R20	;R20 = 1010 1100 C = 0
ROL R20	;R20 = 0101 1000 C = 1

LSL - Logical Shift Left

LSL Rd ;logical shift left

In LSL, as bits are shifted from right to left, 0 enters the LSB and the MSB exits to the carry flag. In other words, **in LSL 0 is moved to the LSB, and the MSB is moved to the C.**



This instruction multiplies content of the register by 2 assuming that after LSL the carry flag is not set.

In the next code you can see what happens to 00100110 after running 3 LSL instructions.

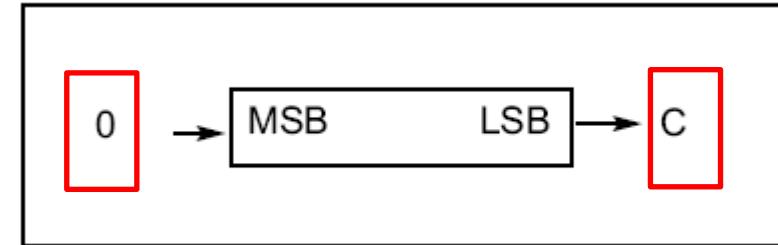
CLC	;make C = 0 (carry is 0)
LDI R20,0x26	;R20 = 0010 0110(38) C = 0
LSL R20	;R20 = 0100 1100(76) C = 0
LSL R20	;R20 = 1001 1000(152) C = 0
LSL R20	;R20 = 0011 0000(48) C = 1 as C=1 and content of R20 ;is not multiplied by 2

LSR - Logical Shift Right

LSR Rd

;Rd (only flags are set)

In LSR, as bits are shifted from left to right, 0 enters the MSB and the LSB exits to the carry flag. In other words, **in LSR 0 is moved to the MSB, and the LSB is moved to the C.**



This instruction divides content of the register by 2 and carry flag contains the remainder of division.

In the next code you can see what happens to 0010 0110 after running 3 LSL instructions.

LDI R20,0x26
LSR R20
LSR R20
LSR R20

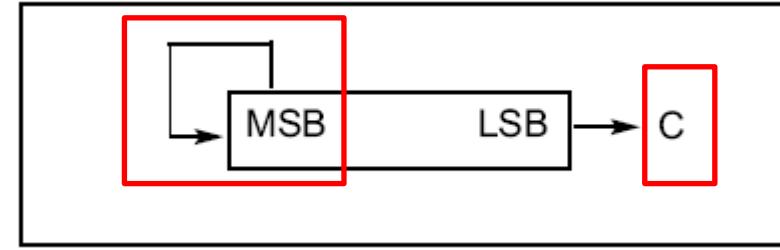
;R20 = 0010 0110 (38)
;R20 = 0001 0011 (19) C = 0
;R20 = 0000 1001 (9) C = 1
;R20 = 0000 0100 (4) C = 1

ASR - Arithmetic Shift Right

ASR Rd ;Rd (only flags are set)

ASR instruction can divide signed number by 2. In LSR, as bits are shifted from left to right, MSB is held constant and the LSB exits to the carry flag. In other words

**MSB is not changed but is copied to D6,
D6 is moved to D5, D5 is moved to D4
and so on.**



In the next code you can see what happens to 0010 0110 after running 5 ASL instructions.

LDI R20 , 0D60	;R20 = 1101 0000(-48) c = 0
ASR R20	;R20 = 1110 1000(-24) C = 0
ASR R20	;R20 = 1111 0100(-12) C = 0
ASR R20	;R20 = 1111 1010(-6) C = 0
ASR R20	;R20 = 1111 1101(-3) C = 0
ASR R20	;R20 = 1111 1110(-1) C = 1

Test ("socrative.com"): Room = AUQUIZ

Hvordan kan man (hurtigt) dividere R16 med 2?
R16 indeholder et positivt tal.

A: ROR R16

B: ROL R16

C: LSR R16

D: LSL R16

E: DIV R16,2

Test ("socrative.com"): Room = AUQUIZ

Hvordan kan man (hurtigt) gange R16 med 2?
R16 indeholder et positivt tal.

A: ROR R16

B: ROL R16

C: LSR R16

D: LSL R16

E: ASR R16

LAB5, del 1

MSB (bit 31)	Tal 1:			LSB (bit 0)
R19	R18	R17	R16	

MSB (bit 31)	Tal 2:			LSB (bit 0)
R23	R22	R21	R20	

MSB (bit 31)	Sum efter addition:			LSB (bit 0)
R19	R18	R17	R16	

1 milliard + 2 milliarder = ?

LAB5, del 2

- Hvis trykknap SW7 aktiveres:
Talværdien på PORTB inkrementeres.
- Hvis trykknap SW6 aktiveres:
Talværdien på PORTB dekrementeres.
- Hvis trykknap SW5 aktiveres:
Værdien på [LED7,LED6,LED5,LED4] ”bytter plads med” [LED3,LED2,LED1,LED0].
- Hvis trykknap SW4 aktiveres:
Alle lysdioderne skifter tilstand (fra tændt til slukket og omvendt).
- Hvis trykknap SW3 aktiveres:
Talværdien på PORTB divideres med 8.
- Hvis trykknap SW2 aktiveres:
Talværdien på PORTB divideres med 7
Hint: Se side 141 i lærebogen.
- Hvis trykknap SW1 aktiveres:
LED7 og LED0 slukkes, mens de andre LEDs skal være uændrede.
- Hvis trykknap SW0 aktiveres:
LED7 og LED0 tændes, mens de andre LEDs skal være uændrede.

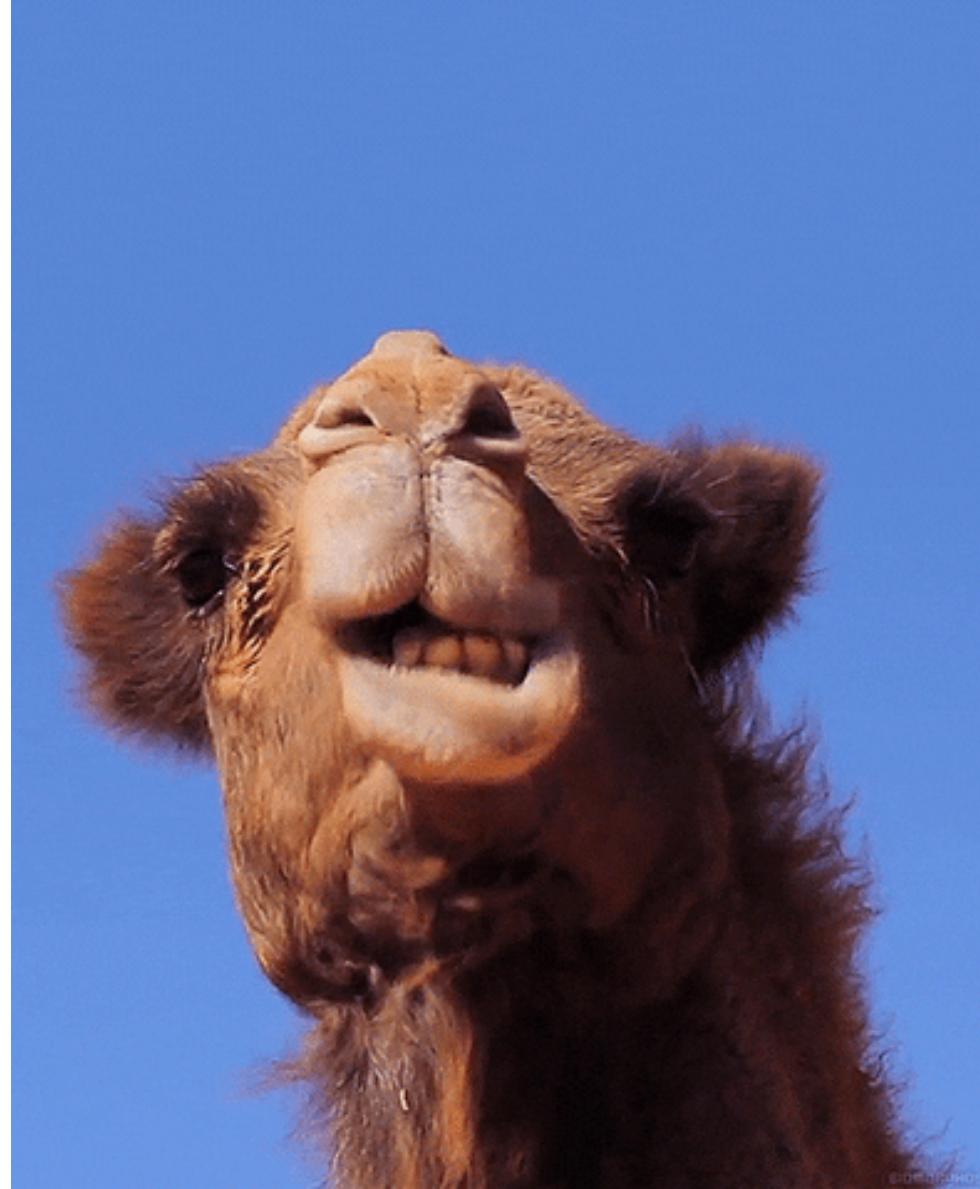
LAB5, del 3 (hvis tid)

```
;***** LED_OFF *****
;**** Slukker en LED på PB ****
;**** Bit nr.(0-7) i R20 ****
;*****
LED_OFF:
    LDI  R21,1          ;R21 = 0b00000001
    CPI  R20,0
    BREQ KLAR1         ;Hop, hvis LED nr. = 0
IGEN1:
    LSL  R21          ;Venstre-skift R21
    DEC  R20          ;ialt "LED nr." pladser
    BRNE IGEN1
KLAR1:
    COM  R21          ;Inverter "masken"
    IN   R20,PINB      ;Aflæs alle LEDs
    AND  R20,R21      ;- lav bitvis AND
    OUT  PORTB,R20    ;- og skriv ud til LEDs igen
    RET
;*****

;***** LED_ON *****
;**** Taender en LED på PB ****
;**** Bit nr.(0-7) i R20 ****
;*****
LED_ON:
    ;----- Skriv den manglende kode her

    RET
;*****
```

Slut

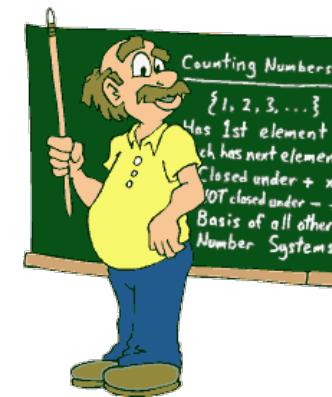


MSYS

Microcontroller Systems

Lektion 10

C programmering



Slides from Henning Hargaard

Michael Alrøe

GCC open source compilers



GCC = Gnu Compiler Collection
Open Source (UNIX/Linux):
☺ Gratis.
☹ Dokumentation kan halte lidt.

AVR GCC: GCC Compiler for Atmel AVR.
Mest udbredt er C-compileren, men C++
compiler findes også.

AVR GCC er integreret i Microchip Studio



Andre C compilers for Atmel AVR (ikke gratis):

- **IAR** (svensk). Meget prof. Dyr licens.
- **CodeVision** (rumænsk). Mindre dyr licens.

Data typer i C

<u>TypeDefs</u>	<u>Data Type</u>	<u>Size in Bits</u>	<u>Data Range/Usage</u>
<i>uint8_t</i>	unsigned char	8-bit	0 to 255
<i>int8_t</i>	char	8-bit	-128 to +127
<i>uint16_t</i>	unsigned int	16-bit	0 to 65,535
<i>int16_t</i>	int	16-bit	-32,768 to +32,767
<i>uint32_t</i>	unsigned long	32-bit	0 to 4,294,967,295
<i>int32_t</i>	long	32-bit	-2,147,483,648 to +2,147,483,648
	float	32-bit	$\pm 1.175\text{e-}38$ to $\pm 3.402\text{e}38$
	double	32-bit	$\pm 1.175\text{e-}38$ to $\pm 3.402\text{e}38$

Brug altid **mindst mulige type** (helst *uint8_t*) !

Hvorfor?

Declaring identifiers in C

Function and variable names are identifiers in C

Rules to define identifier name:

- Must **not** start with a digit.
- Can consist of alphabets (**US only**), digits and special symbols like underscore _.
- Blank/spaces are **not** allowed.
- C keywords are **not** allowed.

```
int x;
int a, b, c, d;
char letter;
float the_float;
```

Fra lille til stor type

- Hvis en variabel af en **større type** sættes lig med en variabel af **mindre type**, sættes automatisk 0'er ind foran.
- Den "lille variabel" opfattes altså midlertidigt som om den har den "store variabels" type.

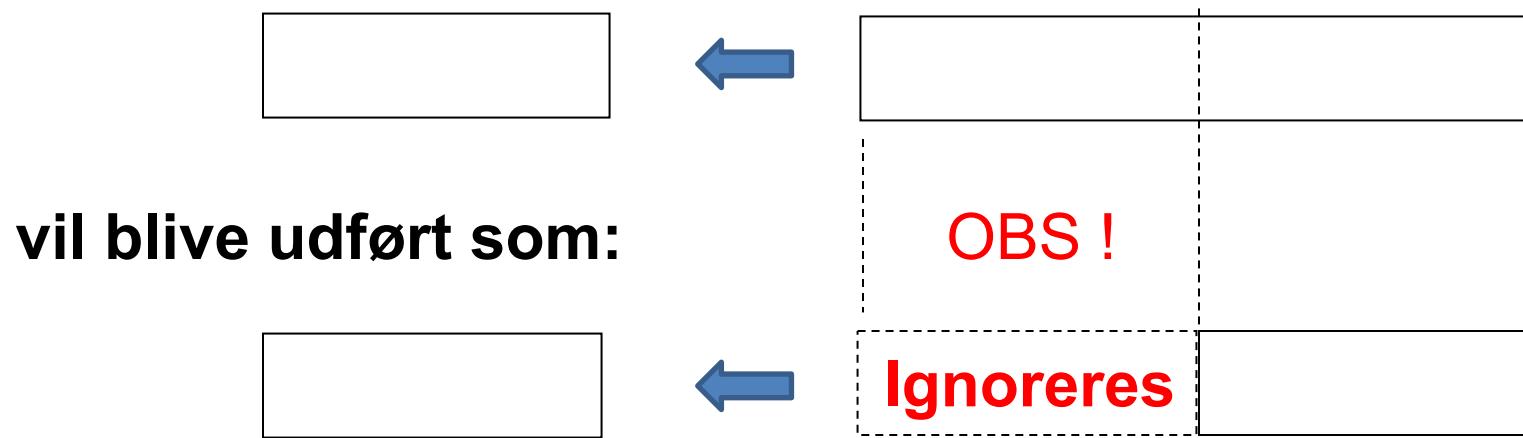


vil blive udført som:



Fra stor til lille type

- Hvis en variabel af en mindre type sættes lig med en variabel af større type, mistes de mest betydende bits !
- Den "store variabel" opfattes altså midlertidigt som om den har den "lille variabels" type.



Eksempler

```
unsigned int x;  
unsigned char y;  
  
// y sættes lig med 200  
y = 200;  
  
// Hvad bliver x nu ?  
x = y;  
  
// x = 12345 = 0x3039  
x = 12345;  
  
// Hvad bliver y nu ?  
y = x;
```

Compiler mellemresultater

- I mange situationer vil compileren lave **"mellemregninger"** – dette gælder specielt, hvor vi skriver en formel i en statement:
Eksempel: $z = (10 * y) + x;$
 x, y og z er erklæret på forhånd.
- Parentesen udregnes først, og her vil compileren have et (midlertidigt) **mellemresultat** af $(10 * y)$.
- Vigtigt: Mellemresultatet vil blive gemt i en **midlertidig variabel af samme type som y !**
- Dette kan give problemer - hvis typen for y er "for lille" til $(10 * y)$.
- Det er ***programmørens (dit!) ansvar*** at sørge for, at dette ikke sker !

Test ("socrative.com"): Room = AUQUIZ

- Hvad bliver z efter følgende kode:

```
unsigned long z;  
unsigned long x = 15000;  
unsigned int y = 63000;  
z = x + (10 * y);
```

- A: 645000
- B: 945630000
- C: 55176

Eksempel

```
unsigned long z;  
unsigned long x = 15000;  
unsigned int y = 63000;  
  
// Hvorfor kan dette gå galt ?  
z = x + (10 * y);
```

- Problemet: Typen for y (16 bit) er for lille til at indeholde mellemresultatet ($10 * y$).
- Hvis y midlertidigt var en unsigned long (32 bit) ville problemet være løst.
- Da vi kun har problemet for "mellemresultatet" kan vi løse problemet via "type casting".

Type casting

- Man laver ”Type Casting” på en variabel ved i parenteser at skrive den ønskede (midlertidige) type foran variablen.

```
unsigned long z;  
unsigned long x = 15000;  
unsigned int y = 63000;  
  
// Nu behandler compileren mellemresultatet som 32 bit  
// - og beregningen bliver korrekt.  
// Men variablen y fylder stadig kun 16 bit|  
z = x + (10 * (unsigned long)y);
```

C og portene

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

int main()
{
    unsigned char z = 0;
    DDRB = 0xFF;           //PB er udgange
    while(1)
    {
        PORTB = z;
        z++;
    }
}
```

Hvad laver programmet ?

16 bit sløjfetæller

```
int main()
{
    unsigned int z = 0;
    DDRB = 0xFF;

    for (z=0; z<50000; z++)
    {
        PORTB = 0b10101010;
        PORTB = 0b01010101;
    }

    while(1)
    {}
}
```

Hvad laver programmet ?

32 bit sløjfetæller

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

int main()
{
    unsigned long z = 0;
    DDRB = 0xFF;

    for (z=0; z<100000; z++)
    {
        PORTB = 0b10101010;
        PORTB = 0b01010101;
    }

    while(1)
    {}
}
```

Hvad laver programmet ?

Time delays i C ("primitiv metode")

- Man kan anvende **for** til at lave en tidsforsinkelse

```
void delay100ms(void){  
    unsigned int i ;  
    for(i=0; i<42150; i++);  
}
```



Hvis du bruger denne metode:

- Tidsforsinkelsen afhænger helt af compileren ☹
- Optimeringsgraden **SKAL** sættes til level 00 ☹

Delays på primitiv vis

Example 7-7

Write an AVR C program to toggle all the bits of Port B continuously with a 100 ms delay. Assume that the system is ATmega 32 with XTAL = 8 MHz.

Solution:

```
#include <avr/io.h>
void delay100ms(void)
{
    unsigned int i;
    for(i=0; i<42150; i++);
}

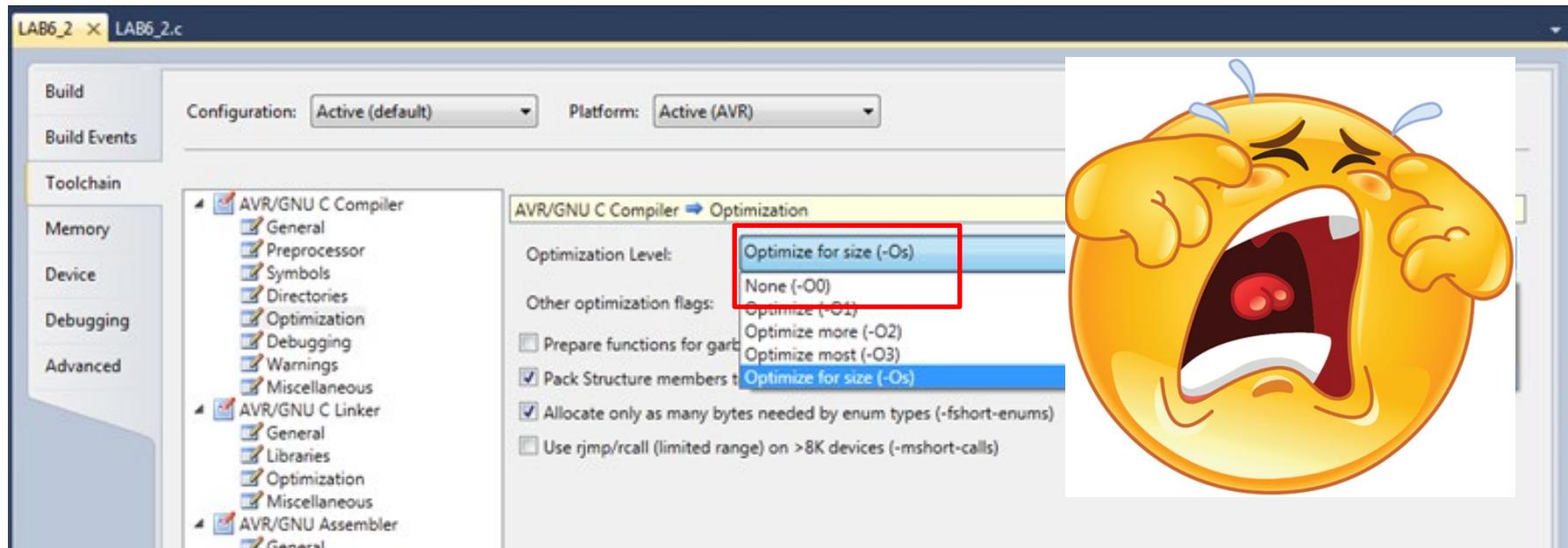
int main(void)
{
    DDRB = 0xFF;
    while (1)
    {
        PORTB = 0xAA;
        delay100ms();
        PORTB = 0x55;
        delay100ms();
    }
    return 0;
}
```



on your
e result.

OBS: Compiler optimering
skal slås fra !
Hvorfor ?

Sådan sættes optimering til “level 00”



Vælg "Properties" for projektet.
Under "Toolchain" → "Optimization" vælges "None".

MEN så genereres in-effektiv kode 😞

Time delay funktioner i C

- BEDRE:

Man kan anvende **pre-definerede** compiler funktioner til at lave tidsforsinkelser.

Start med:

```
#define F_CPU 16000000  
#include <util/delay.h>
```

- og herefter kan man bruge:

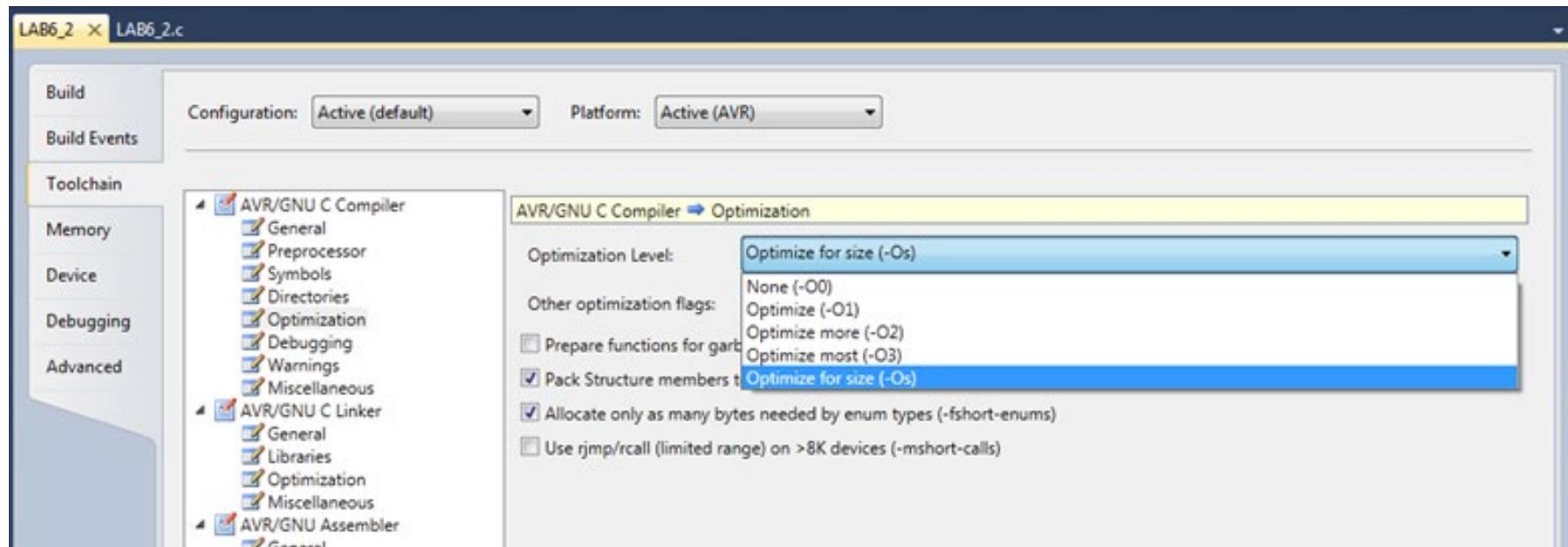
```
_delay_ms(270);  
_delay_us(536);
```

- Metoden er compiler afhængig – ikke hardware afhængig.



Compiler optimization og "delay.h"

OBS: Når man anvender delay-biblioteket <util/delay.h>, må compilerens optimering **IKKE** slås fra (sæt til alt andet end "None").
Det virker ULOGISK, men alligevel sandt.



Logiske operatorer

Table 7-3: Bit-wise Logic Operators for C

		AND	OR	EX-OR	Inverter
A	B	A&B	A B	A^B	$Y = \sim B$
0	0	0	0	0	1
0	1	0	1	1	0
1	0	0	1	1	
1	1	1	1	0	

1110 1111
& 0000 0001

0000 0001

1110 1111
0000 0001
1110 1111

\sim 1110 1011

0001 0100

Logiske operatorer i C

		AND	OR	EX-OR	Inverter
A	B	A&B	A B	A [^] B	Y=~B
0	0	0	0	0	1
0	1	0	1	1	0
1	0	0	1	1	
1	1	1	1	0	

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

int main()
{
    DDRA = 0xFF;
    DDRB = 0xFF;
    DDRC = 0xFF;
    DDRD = 0xFF;

    PORTA = 0b00111111 & PORTA;
    PORTB = 0b00000010 | PORTB;
    PORTC = 0b11110000 ^ PORTC;
    PORTD = ~PORTD;
    while(1)
    {}
}
```

Logiske operatorer

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

int main()
{
    DDRB = 0xFF;
    while(1)
    {
        PORTB = PORTB | 0b00010000;
        PORTB = PORTB & 0b11101111;
    }
}
```

Hvad laver programmet ?

Logiske operatorer

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

int main()
{
    DDRA = 0;
    DDRB = 0xFF;
    while(1)
    {
        if ((PINA & 0b00000010) == 0)
            PORTB = 17;
        else
            PORTB = 117;
    }
}
```

Hvad laver programmet ?

Compound assignment

Operation	Abbreviated Expression	Equal C Expression
And assignment	a &= b	a = a & b
OR assignment	a = b	a = a b

```
#include <avr/io.h>          //standard AVR header
int main(void)
{
    DDRB &= 0b11011111;        //bit 5 of Port B is input
    DDRC |= 0b10000000;       //bit 7 of Port C is output

    while (1)
    {
        if(PINB & 0b00100000)
            PORTC |= 0b10000000; //set bit 7 of Port C to 1
        else
            PORTC &= 0b01111111; //clear bit 7 of Port C to 0
    }
    return 0;
}
```

Bit shifting i C

- `data >> number of bits to be shifted right`
- `data << number of bits to be shifted left`

1110 0000 >> 3

0001 1100

0000 0001 <<2

0000 0100

Bit shifting i C

Operation	Symbol	Format of Shift Operation
Shift right	>>	data >> number of bits to be shifted right
Shift left	<<	data << number of bits to be shifted left

The following shows some examples of shift operators in C:

1. `0b00010000 >> 3 = 0b00000010` /* shifting right 3 times */
2. `0b00010000 << 3 = 0b10000000` /* shifting left 3 times */
3. `1 << 3 = 0b00001000` /* shifting left 3 times */

Hvad svarer `>>` til i assembly ?
Hvad svarer `<<` til i assembly ?

Anvendelse af bit shifting

Example 7-22

Write code to generate the following numbers:

- (a) A number that has only a one in position D7
- (b) A number that has only a one in position D2
- (c) A number that has only a one in position D4
- (d) A number that has only a zero in position D5
- (e) A number that has only a zero in position D3
- (f) A number that has only a zero in position D1

Solution:

- (a) `(1<<7)`
- (b) `(1<<2)`
- (c) `(1<<4)`
- (d) `~(1<<5)`
- (e) `~(1<<3)`
- (f) `~(1<<1)`

Sådan sættes en bit i en byte til 1

- Vi kan bruge | operatoren til at sætte en bit i en byte til 1 :

xxxx xxxx

| 0001 0000

xxx1 xxxx

ELLER

xxxx xxxx

| 1 << 4

xxx1 xxxx

```
PORTB |= ( 1 << 4 ); //Set bit 4 (5th bit) of PORTB
```

Sådan nulstilles en bit i en byte til 0

- Vi kan bruge & operatoren til at nulstille en bit i en byte til 0 :

	xxxx xxxx
&	1110 1111

	xxx0 xxxx

ELLER

	xxxx xxxx
&	$\sim(1 \ll 4)$

	xxx0 xxxx

```
PORTB &= ~ ( 1 << 4 ); //Clear bit 4 (5th bit) of PORTB
```

Sådan checkes en bit i en byte

- Vi kan bruge & operatoren til at se, om en bit i en byte er 1 eller 0 :

	xxxx xxxx
&	0010 0000

	00x0 0000

ELLER

	xxxx xxxx
&	(1 << 5)

	00x0 0000

```
if (PINC & (1 << 5)) // check bit 5 (6th bit) of PINC
```

HUSK, for C gælder reglen :

- "Alt, der er nul" = FALSE.
- "Alt, der er forskellig fra nul" = TRUE.

Bit shifting

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

int main()
{
    DDRB = 0xFF;
    while(1)
    {
        PORTB |= (1<<4);
        PORTB &= ~(1<<4);
    }
}
```

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

int main()
{
    DDRB = 0xFF;
    while(1)
    {
        PORTB = PORTB | 0b00010000;
        PORTB = PORTB & 0b11101111;
    }
}
```

OBS: Det er blot forskellige måder at skrive C-koden på.
Maskinkoden bliver den samme !

#define for bit numre

Example 7-25

A door sensor is connected to the port B pin 1, and an LED is connected to port C pin 7. Write an AVR C program to monitor the door sensor and, when it opens, turn on the LED.

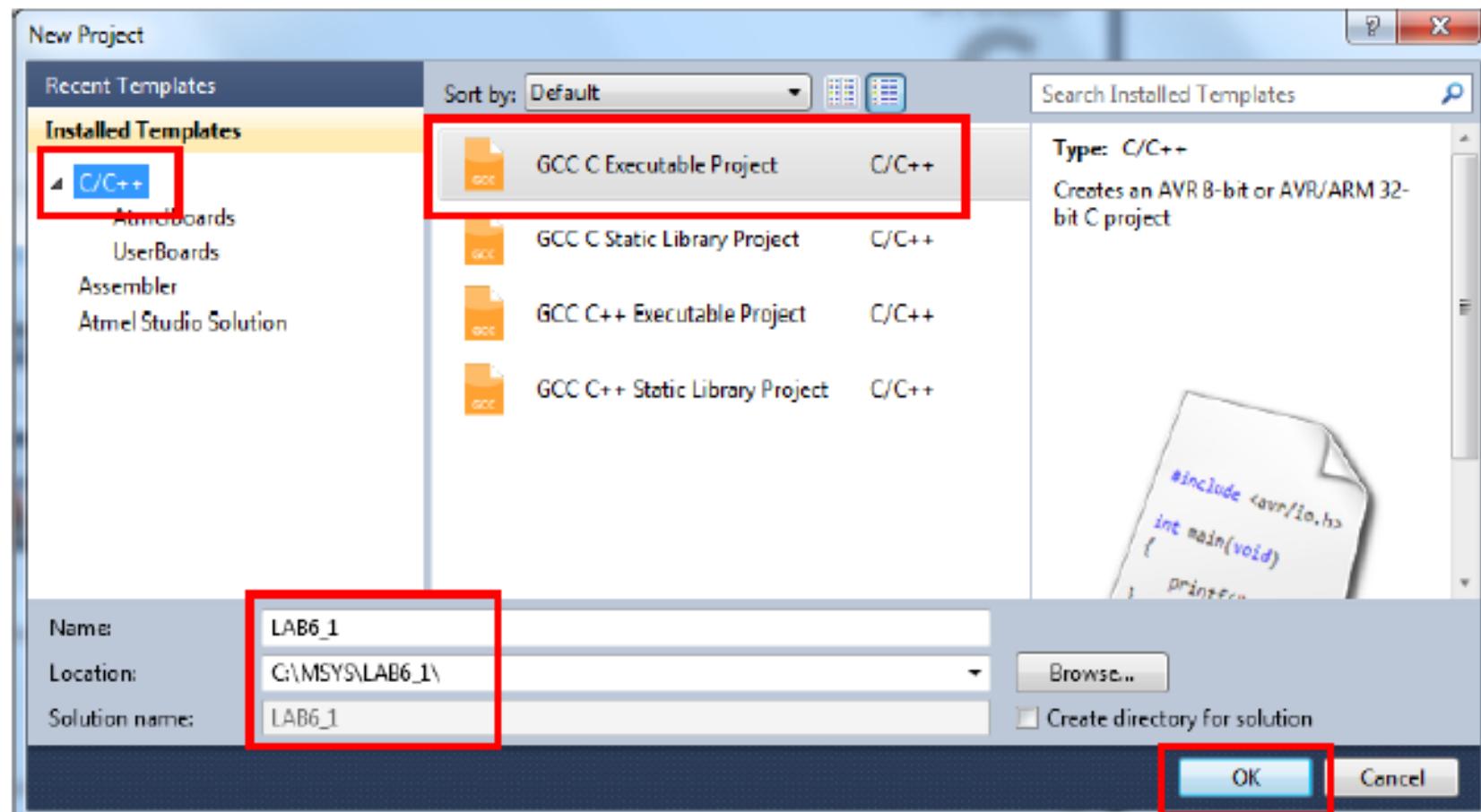
Solution:

```
#include <avr/io.h>                                //standard AVR header
#define LED 7
#define SENSOR 1

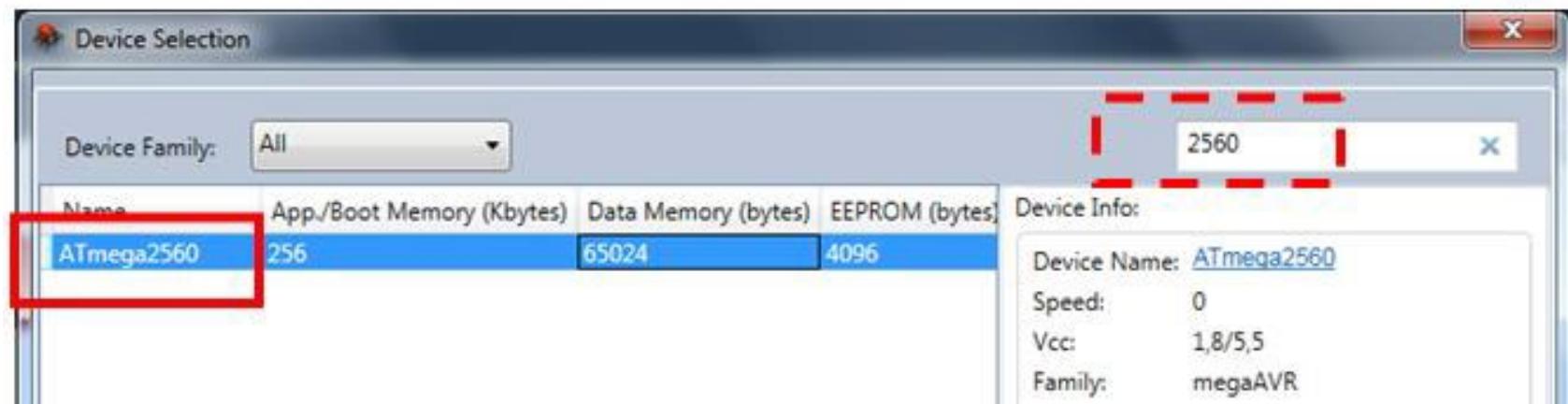
int main(void)
{
    DDRB = DDRB & ~(1<<SENSOR);          //SENSOR pin is input
    DDRC = DDRC | (1<< LED);           //LED pin is output

    while(1)
    {
        if (PINB & (1 << SENSOR))       //check SENSOR pin of PINB
            PORTC = PORTC | (1<<LED);   //set LED pin of Port C
        else
            PORTC = PORTC & ~(1<<LED); //clear LED pin of Port C
    }
    return 0;
}
```

AVR GCC C projekt i Microchip Studio



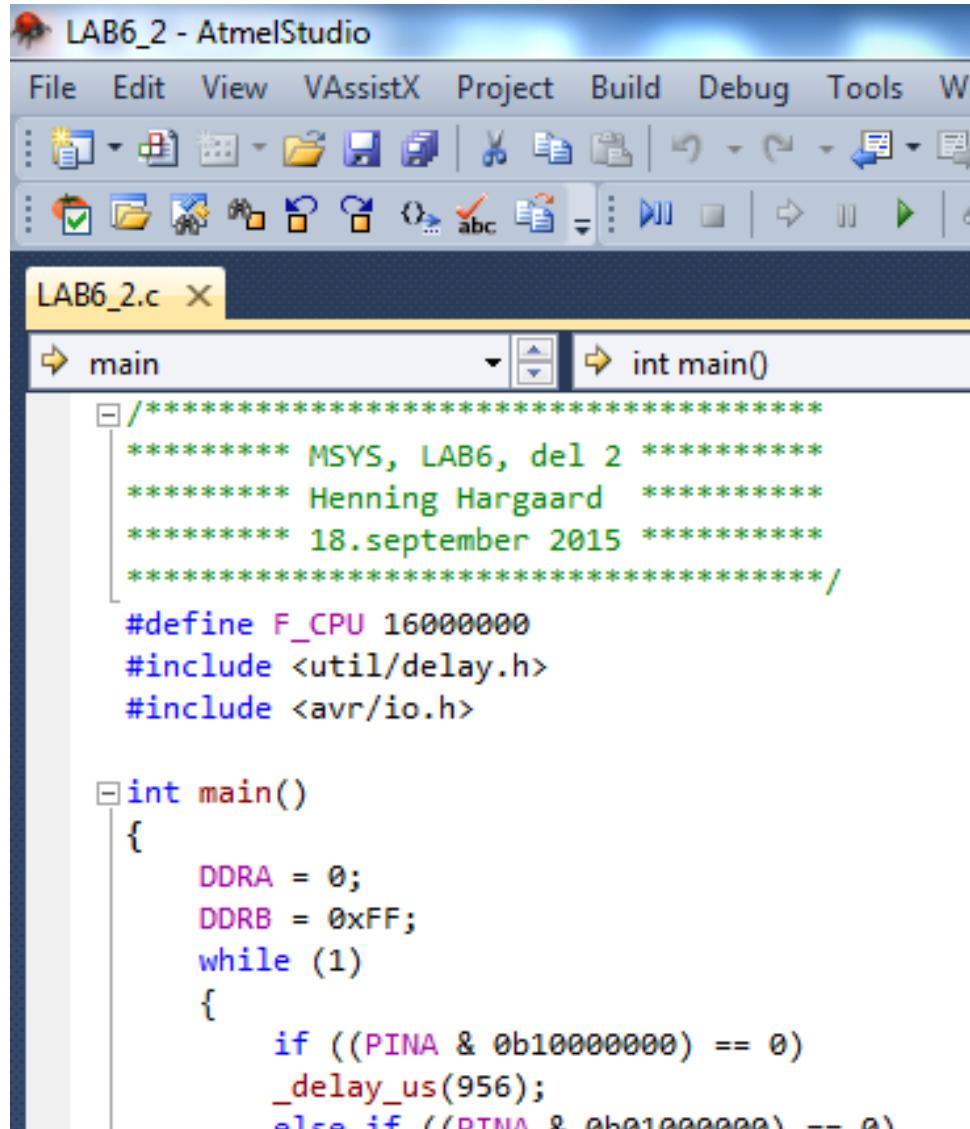
ACR GCC C projekt i Microchip Studio



The code editor window displays the file 'LAB6_1.c'. The code is as follows:

```
/*  
 * LAB6_1.c  
 *  
 * Created: 18-09-2015 14:13:03  
 * Author: hh  
 */  
  
#include <avr/io.h>  
  
int main(void)  
{  
    while(1)  
    {  
        //TODO:: Please write your application code  
    }  
}
```

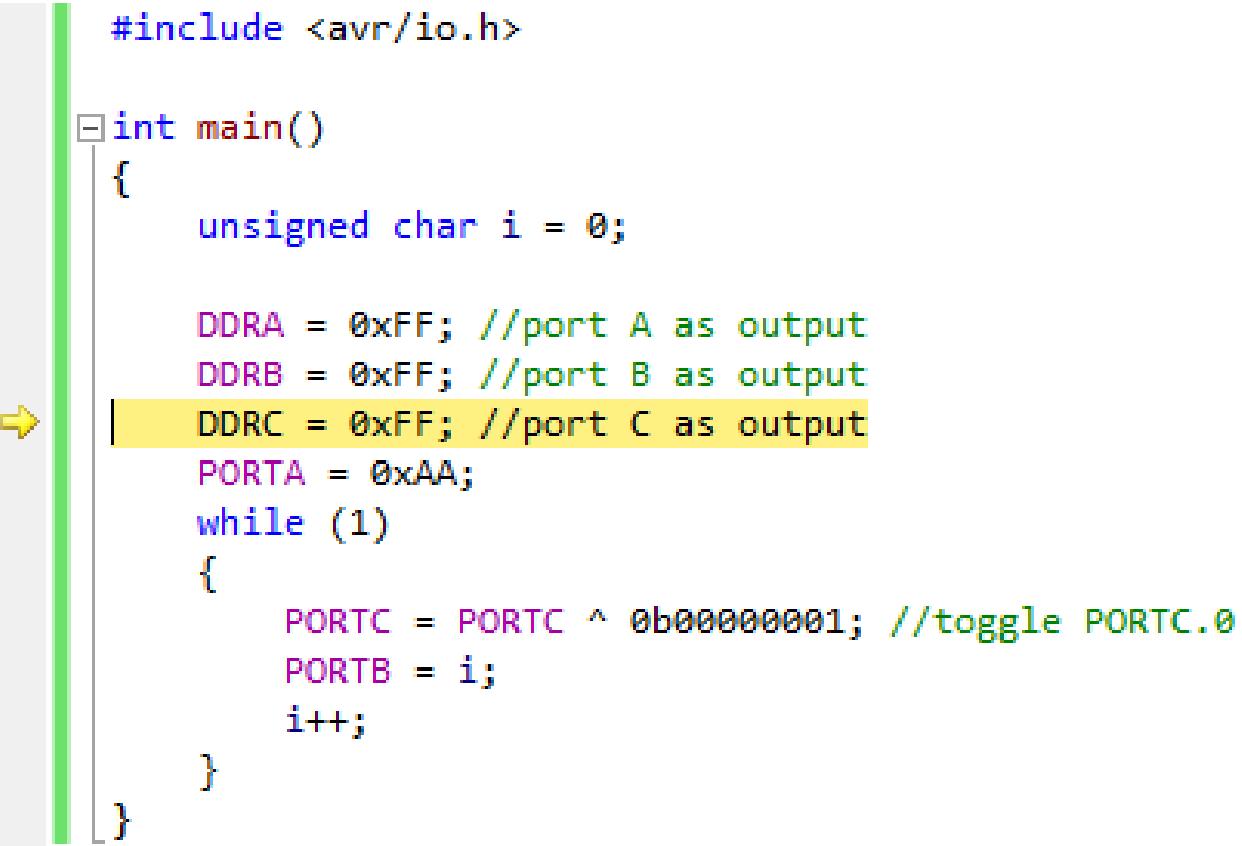
ACR GCC C projekt i Microchip Studio



```
LAB6_2 - AtmelStudio
File Edit View VAssistX Project Build Debug Tools Window Help
LAB6_2.c X
main int main()
{
    /*****
     * MSYS, LAB6, del 2
     * Henning Hargaard
     * 18.september 2015
     *****/
#define F_CPU 16000000
#include <util/delay.h>
#include <avr/io.h>

int main()
{
    DDRA = 0;
    DDRB = 0xFF;
    while (1)
    {
        if ((PIN_A & 0b10000000) == 0)
            _delay_us(956);
        else if ((PIN_A & 0b10000000) == 0x01)
            _delay_ms(1);
    }
}
```

Man kan også debugge på C-niveau

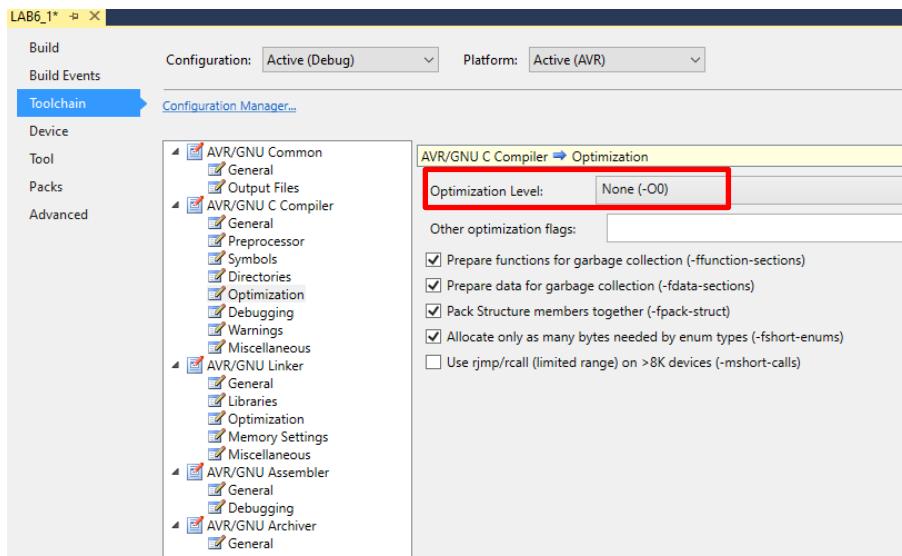
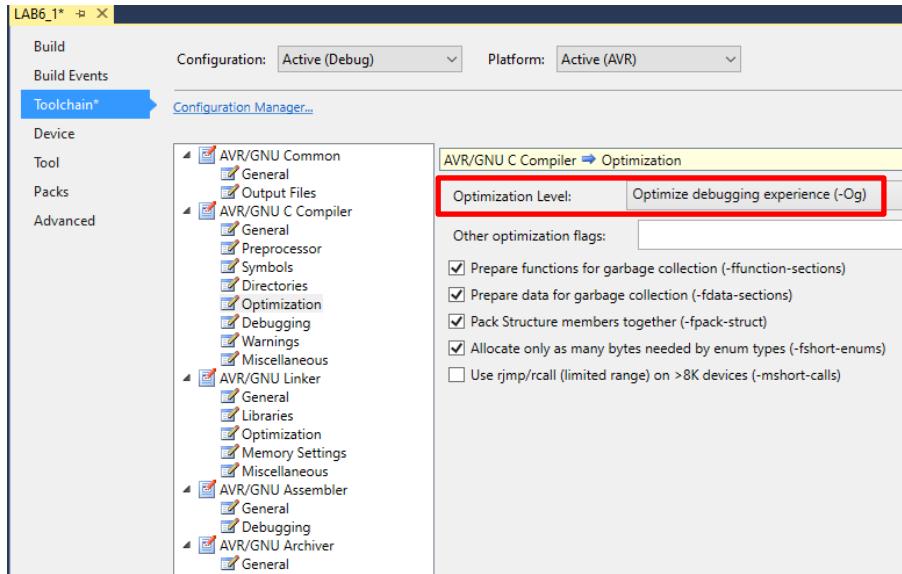


```
#include <avr/io.h>

int main()
{
    unsigned char i = 0;

    DDRA = 0xFF; //port A as output
    DDRB = 0xFF; //port B as output
    DDRC = 0xFF; //port C as output
    PORTA = 0xAA;
    while (1)
    {
        PORTC = PORTC ^ 0b00000001; //toggle PORTC.0
        PORTB = i;
        i++;
    }
}
```

Før debug: Husk Optimazation level



Under "Project ->
Properties ->
Toolchain ->
C compiler Optimazation

vælges

"Optimize debugging
experience" (bedst)
eller "None".

Øvelsen

I denne øvelse vil vi:

- Del 1: Skrive et simpelt C program og simulere afviklingen via Microchip Studios debugger.
- Del 2: Skrive et C program, der anvender de 8 trykknapper på "Mega2560 I/O shield" og en lydgiver til at spille en C dur skala (samme funktionalitet som i en tidligere LAB øvelse).
- Del 3: Skrive et C program, der anvender de 8 trykknapper og en lysdiode på "Mega2560 I/O shield" til at regulere lysdiodens lysintensitet ved hjælp af PWM (samme funktionalitet som i en tidligere LAB øvelse).

Øvelsen, del 1

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

int main()
{
    unsigned char i = 0;

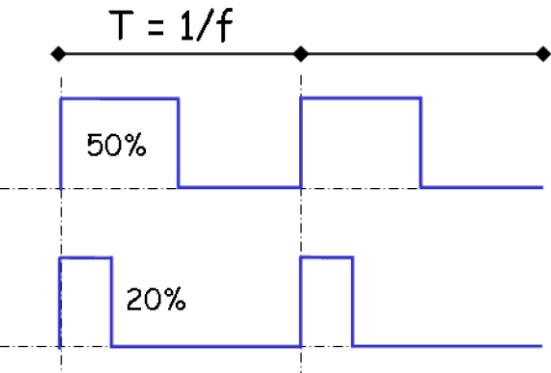
    DDRA = 0xFF; //port A as output
    DDRB = 0xFF; //port B as output
    DDRC = 0xFF; //port C as output
    PORTA = 0xAA;
    while (1)
    {
        PORTC = PORTC ^ 0b00000001; //toggle PORTC.0
        PORTB = i;
        i++;
    }
}
```

Øvelsen, del 2



1. Konfigurer PA benene til indgange (PA er forbundet til trykknapperne).
 2. Konfigurer PB benene til udgange (højtaleren er forbundet til PORTB bit 5).
- Punkt 3 og 4 nedenunder gentages uendeligt (brug en while(1) til det):
3. Ved hjælp af 8 if-sætninger undersøges, om der er trykket på hver enkelt af de 8 taster.
Hvis det er tilfældet, laves en tidforsinkelse svarende til tonens halve periodetid (= T/2).
 4. Hele PINA registeret aflæses for at finde ud af, om nogen taster er aktiveret (brug en if-sætning).
Hvis det er tilfældet, inverteres PORTB (brug C's ~ operator til det).

Øvelsen, del 3



- Mens knappen **SW0** trykkes ned, skal lysdioden styres med et PWM signal, der har en periodetid på **100 ms** og **5%** duty cycle (lysdioden tændt 5 % af tiden).
- Mens knappen **SW1** trykkes ned, skal lysdioden styres med et PWM signal, der har en periodetid på **100 ms** og **95%** duty cycle (lysdioden tændt 95 % af tiden).
- Mens knappen **SW2** trykkes ned, skal lysdioden styres med et PWM signal, der har en periodetid på **1 ms** og **5%** duty cycle (lysdioden tændt 5 % af tiden).
- Mens knappen **SW3** trykkes ned, skal lysdioden styres med et PWM signal, der har en periodetid på **1 ms** og **95%** duty cycle (lysdioden tændt 95 % af tiden).
- Når ingen af ovenstående knapper er trykket ned, skal lysdioden være slukket.

Slut

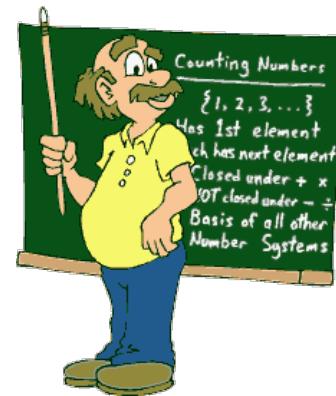


MSYS

Microcontroller Systems

Lektion 11

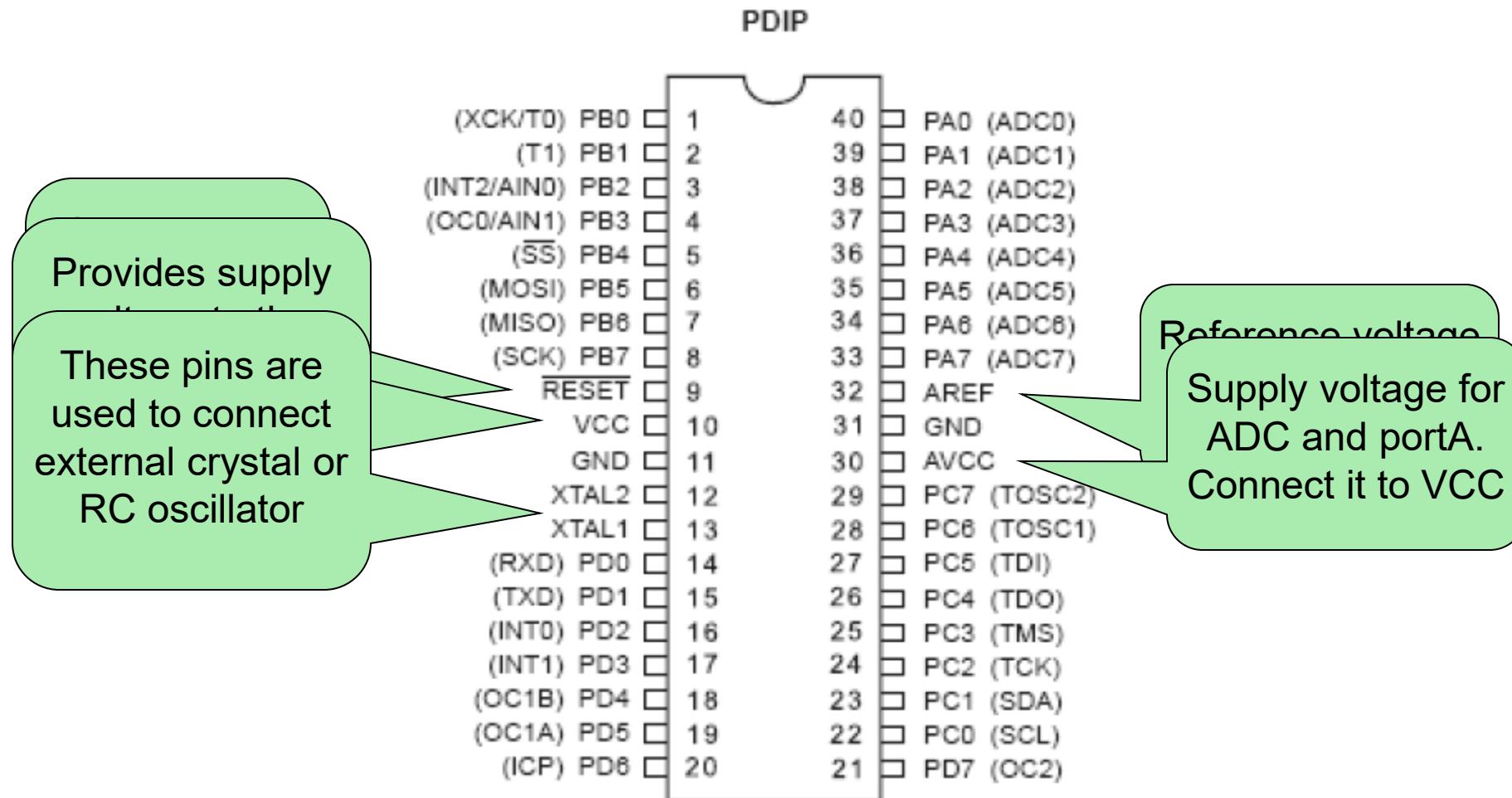
AVR hardware konfiguration



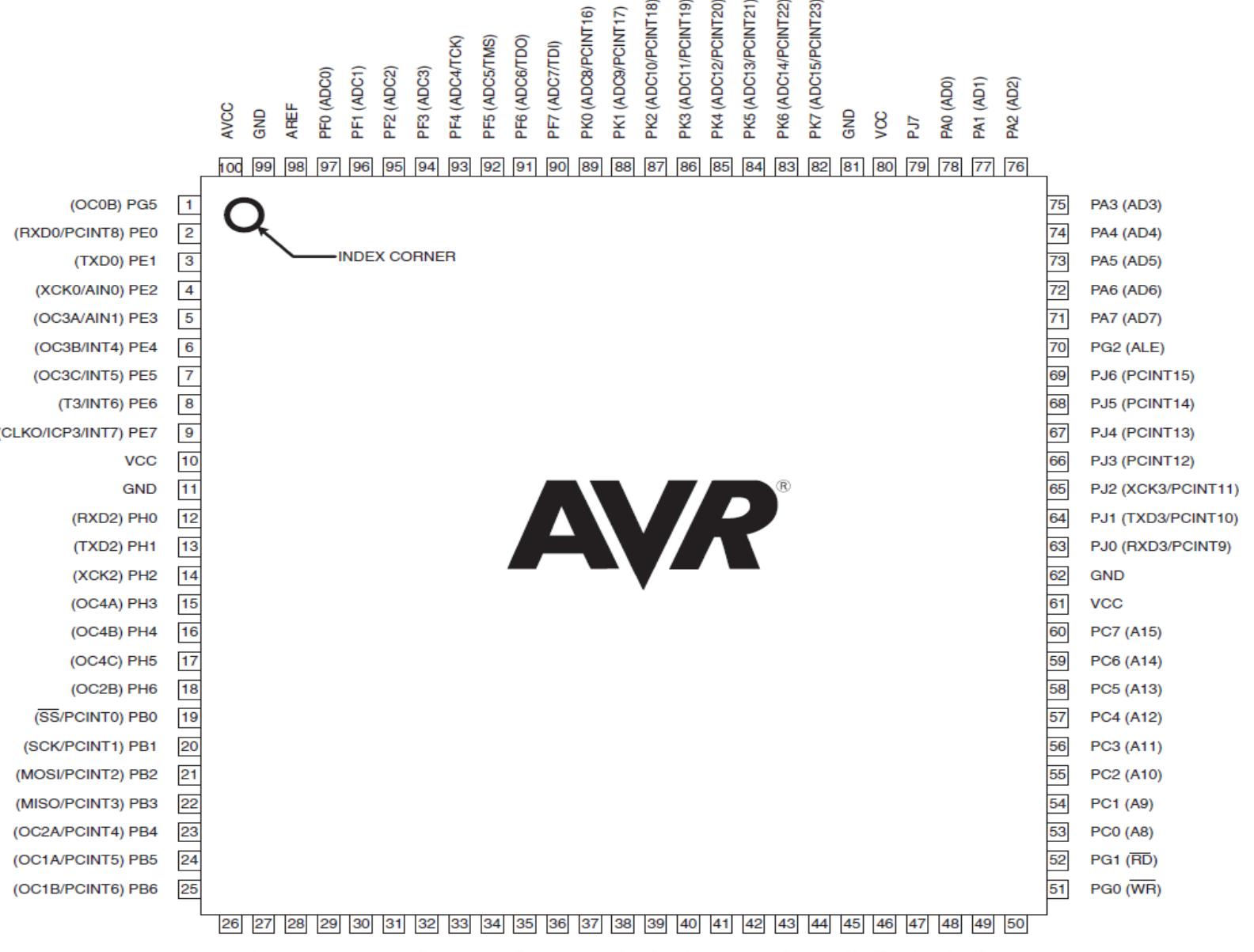
Slides from Henning Hargaard

Michael Alrøe

Mega32 pins

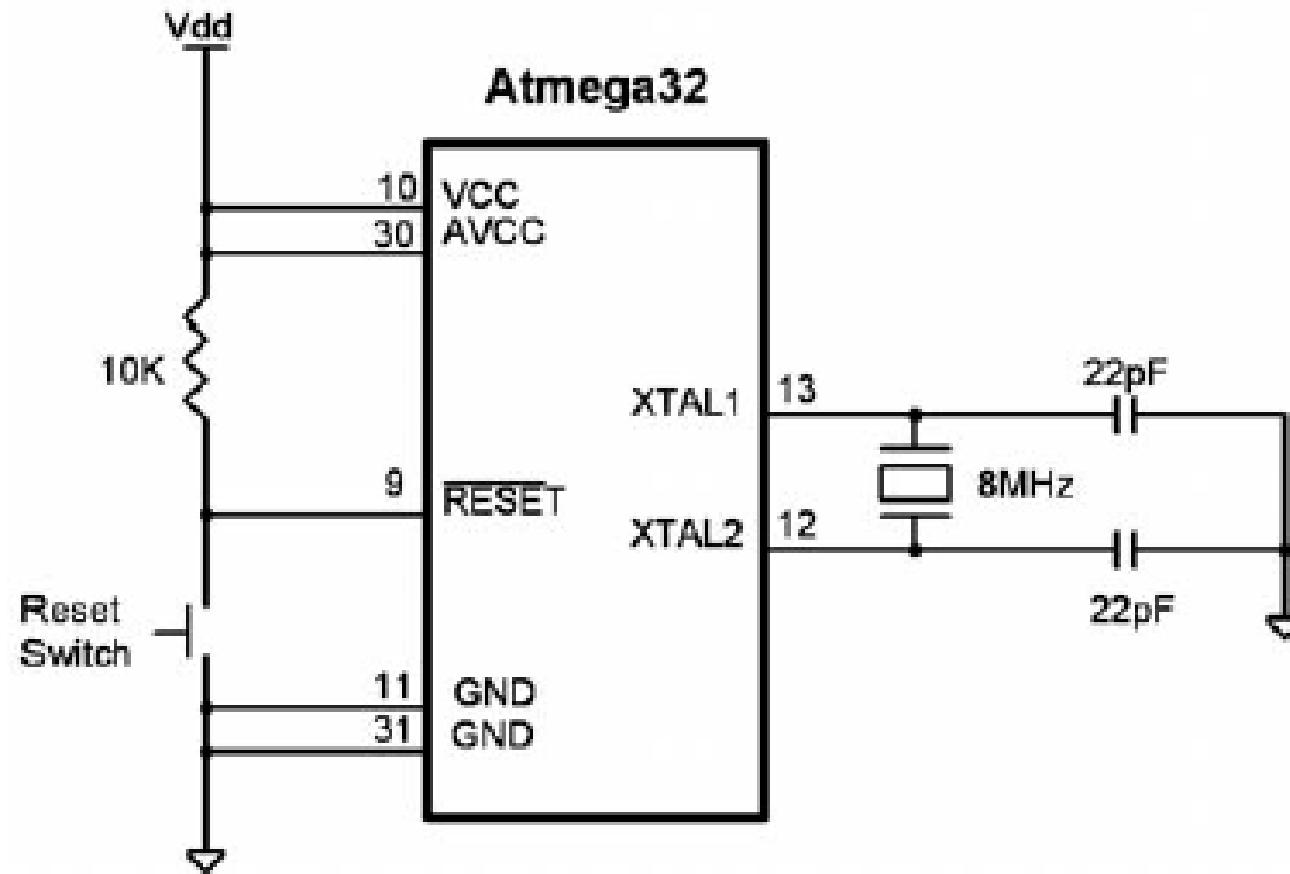


Mega2560 pins

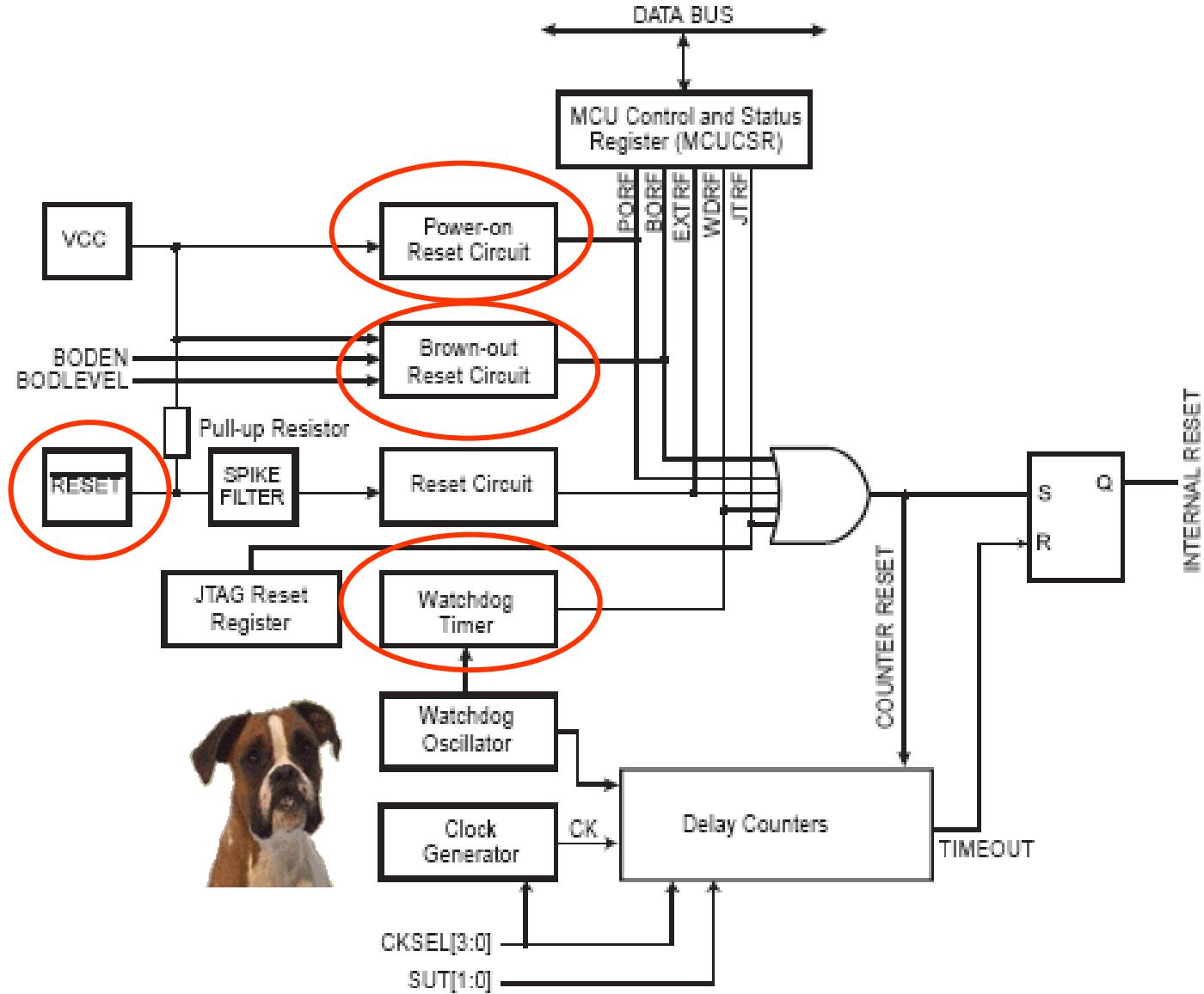


AVR[®]

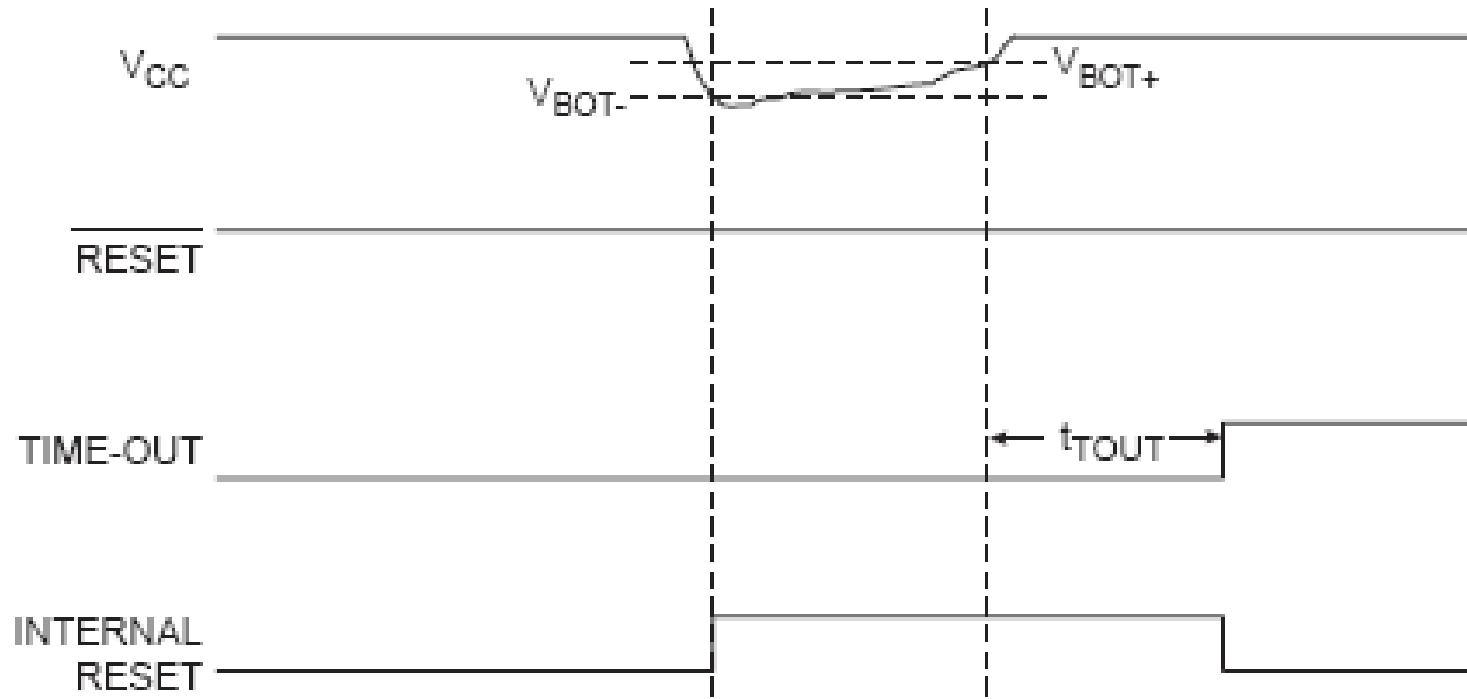
Simpel HW konfiguration



RESET logik



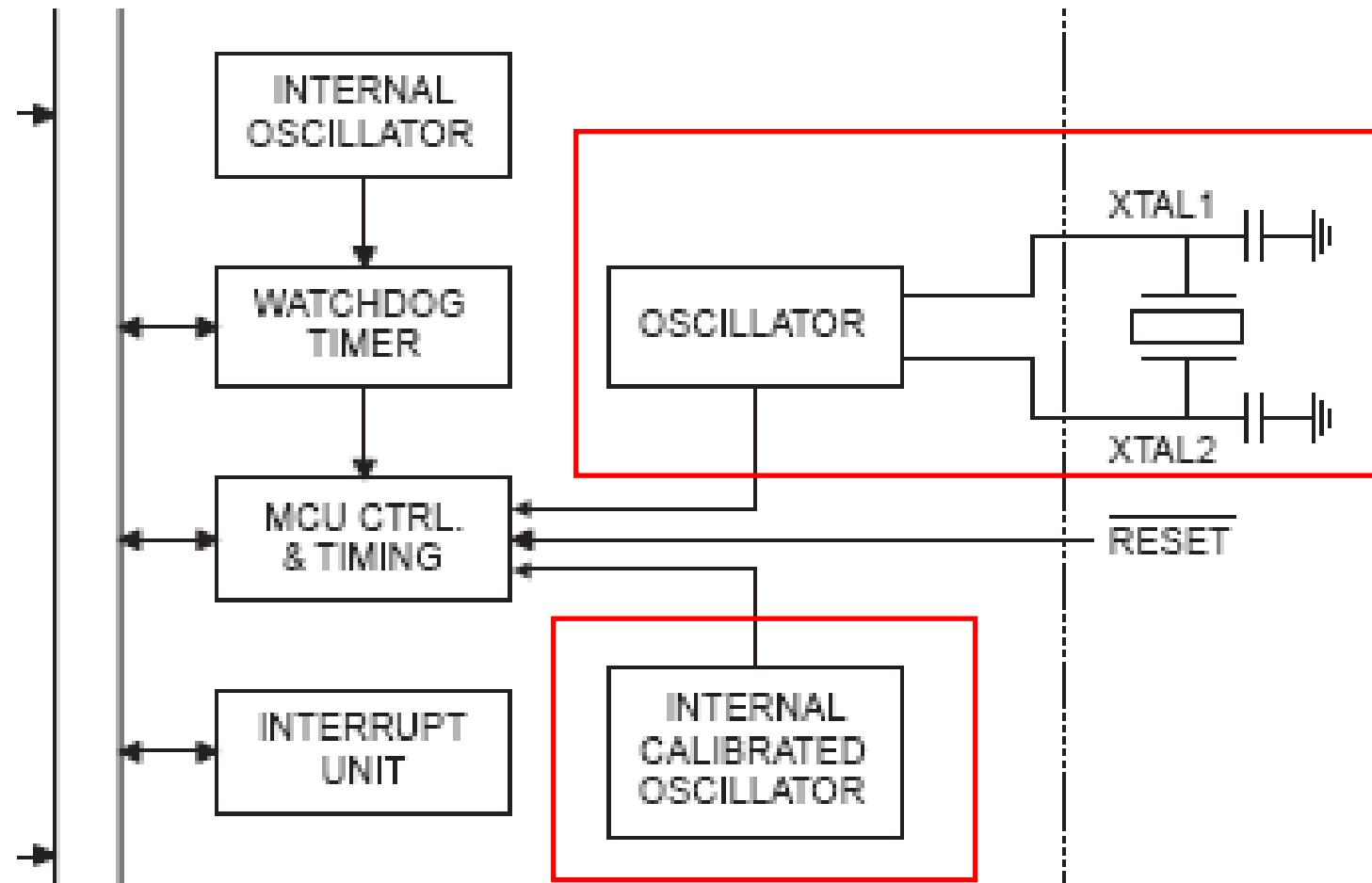
Brown Out RESET



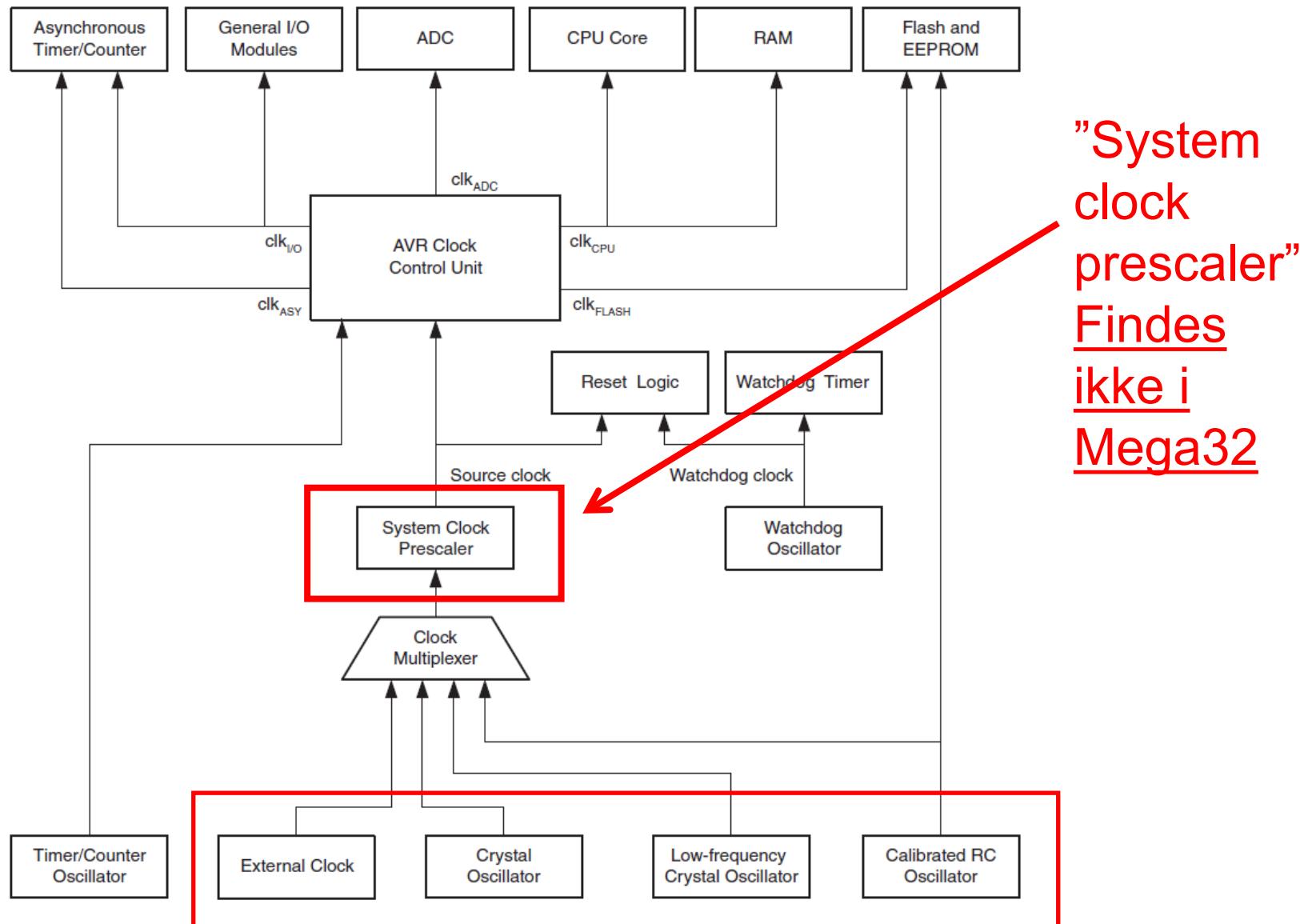
OBS:

Brown out RESET kun aktiv, hvis enabled af programmøren

Clock options



Mega2560: Intern clock distribution



CLKPR = System Clock Prescale Register

7	6	5	4	3	2	1	0	
CLKPCE	-	-	-	CLKPS3	CLKPS2	CLKPS1	CLKPS0	CLKPR
R/W	R	R	R	R/W	R/W	R/W	R/W	
0	0	0	0					See Bit Description

CLKPS3	CLKPS2	CLKPS1	CLKPS0	Clock Division Factor
0	0	0	0	1
0	0	0	1	2
0	0	1	0	4
0	0	1	1	8
0	1	0	0	16
0	1	0	1	32
0	1	1	0	64
0	1	1	1	128
1	0	0	0	256

Metode (sæt prescaler til 32):

CLKPR = 0b10000000; //Skal være først

CLKPR = 0b00000101; //Straks derefter

Clock "fuses"

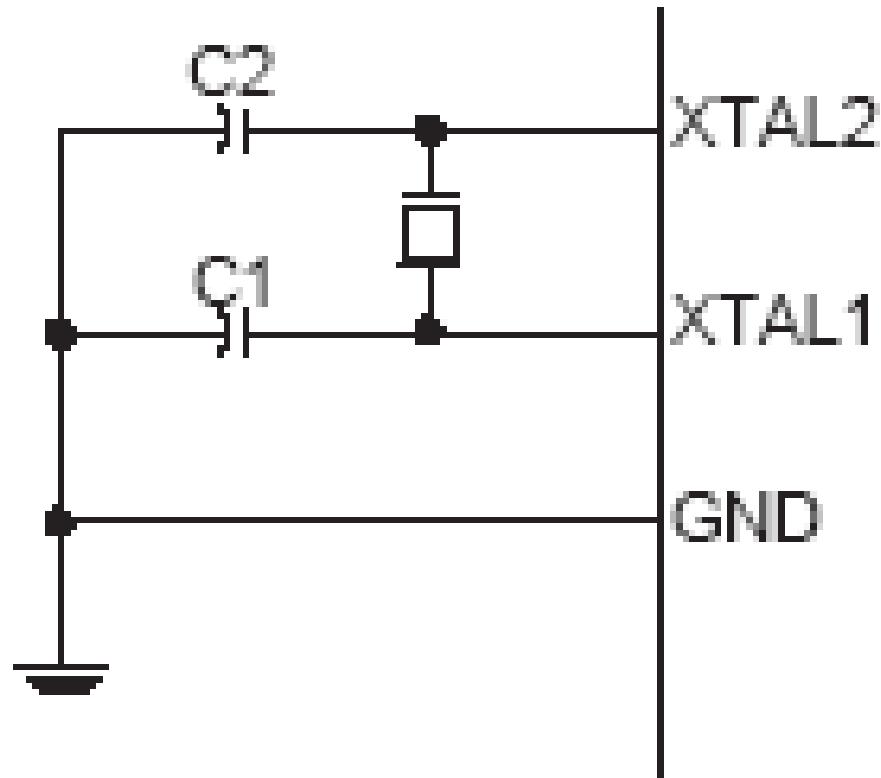
Table 2. Device Clocking Options Select⁽¹⁾

Device Clocking Option	CKSEL3..0
External Crystal/Ceramic Resonator	1111 - 1010
External Low-frequency Crystal	1001
External RC Oscillator	1000 - 0101
Calibrated Internal RC Oscillator	0100 - 0001
External Clock	0000

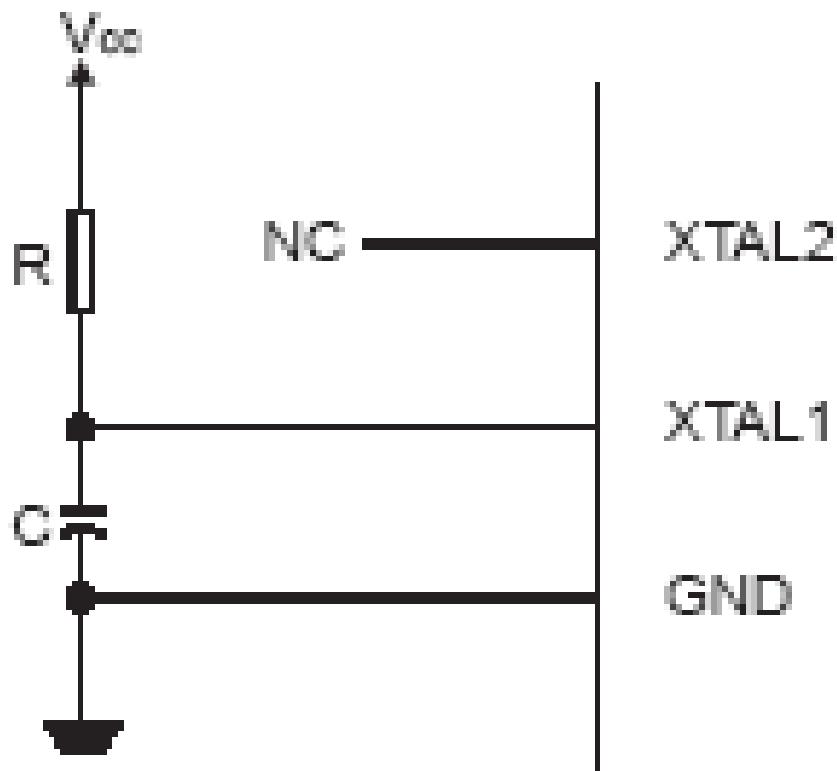
Note: 1. For all fuses "1" means unprogrammed while "0" means programmed.

- Valg af clock source sker med et programmeringsværktøj (f.eks. "AVRdude").
- Vores "Arduino Mega2560" board er sat op til at anvende et eksternt krystal - men det kan vi ændre (anbefales ikke).

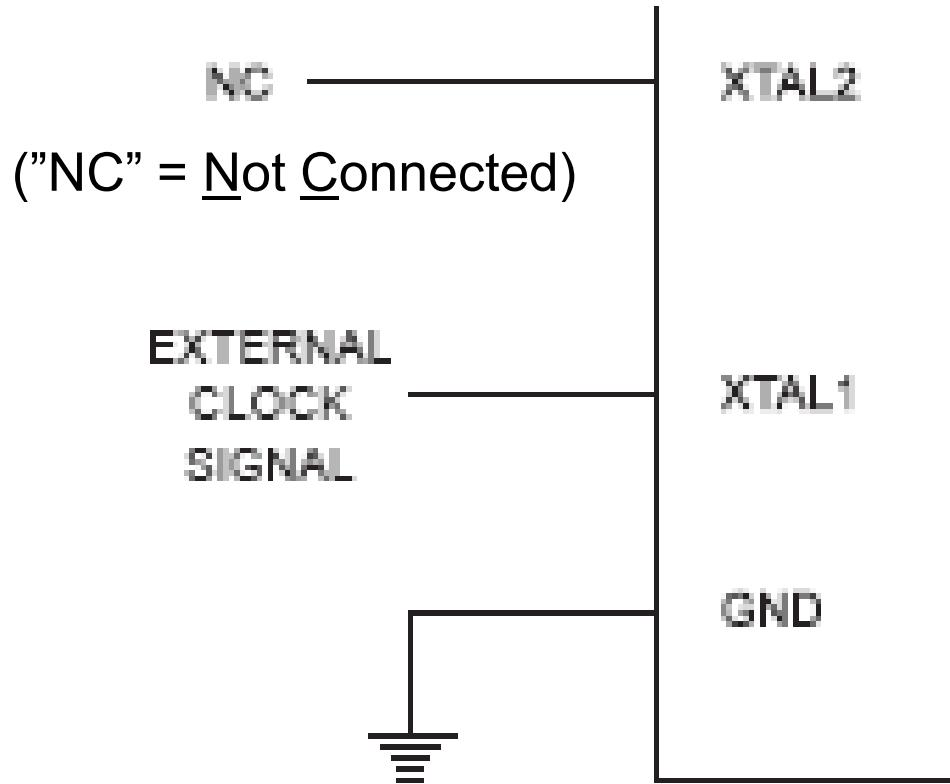
Hvis krystal oscillator vælges:



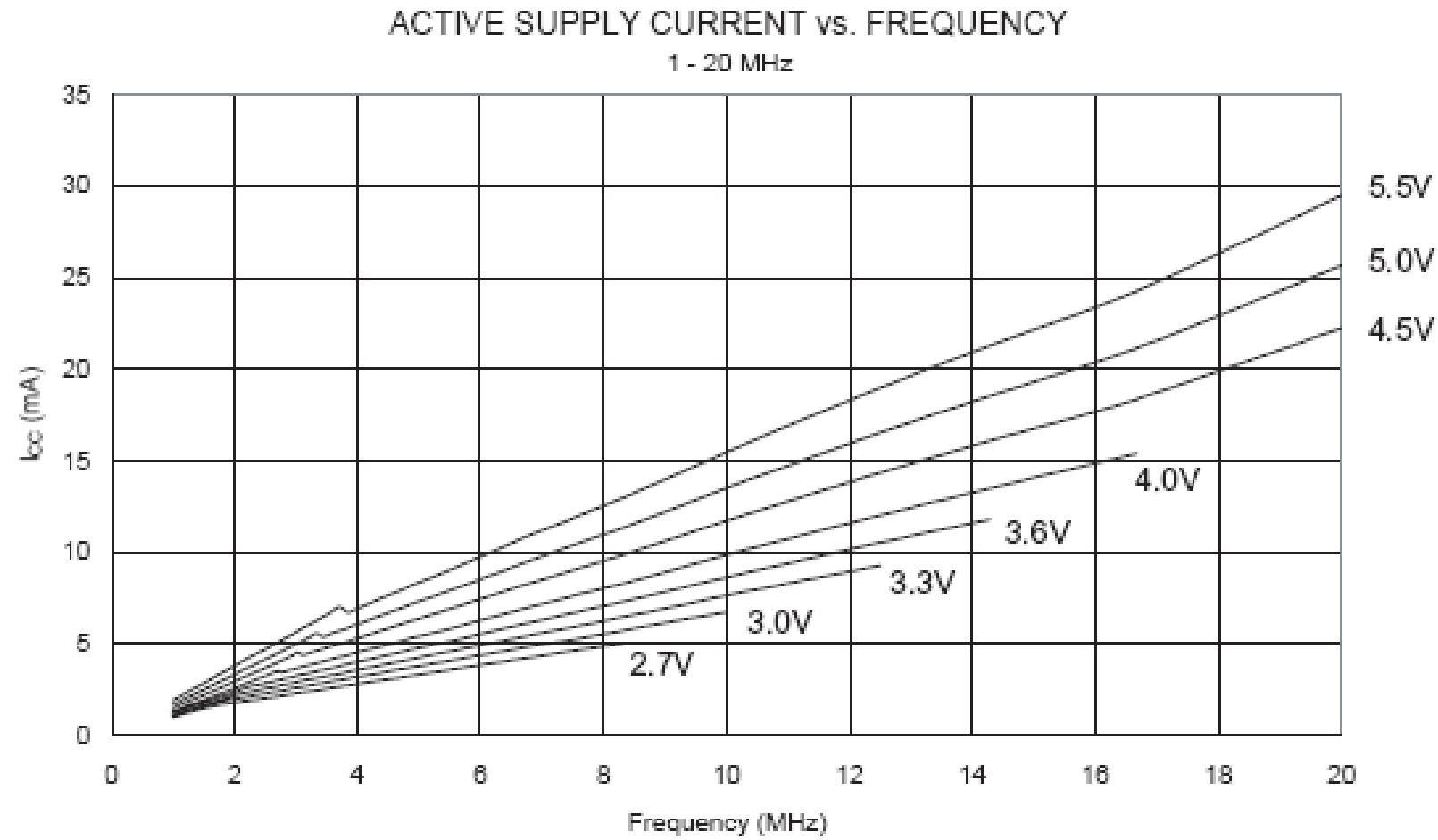
Hvis ekstern RC-oscillator vælges:



Hvis ekstern clocksignal vælges:



Strømforbrug vs. frekvens



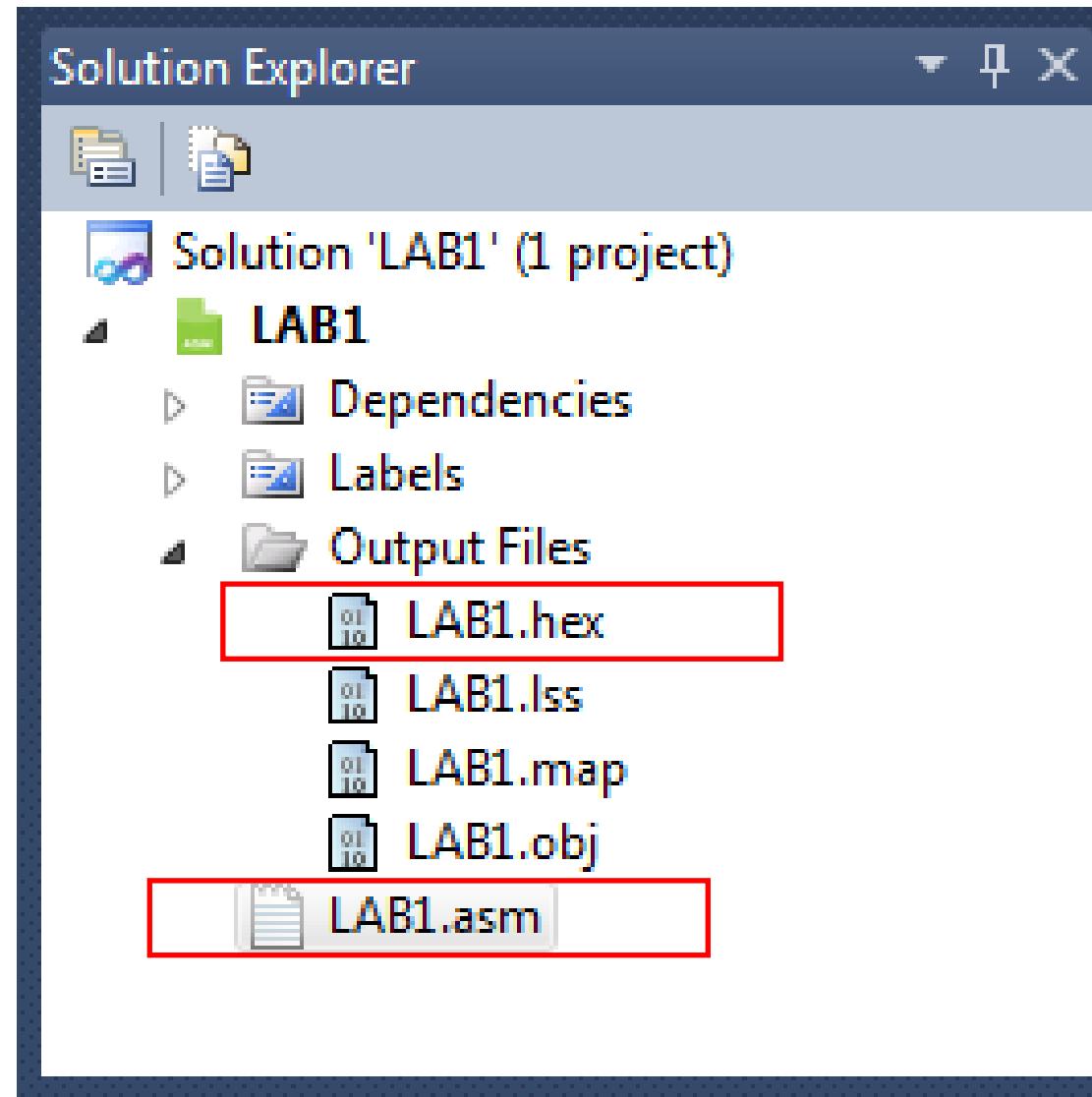
Test ("socrative.com"): Room = AUQUIZ

- Vi har valgt clock fuses til "external crystal" og anvender et krystal på 1 MHz.
Hvad bliver CPU clockfrekvensen efter :
 $\text{CLKPR} = 0b10000000;$
 $\text{CLKPR} = 0b00001000;$
- A: 1 MHz
B: 3,9 kHz
C: 62,5 kHz
D: 256 kHz

Uddrag af en .hex - fil

```
:0C000000C942B000C9400000C940000E9
:10000C000C9400000C9400000C9400000C94000064
:10001C000C9400000C9400000C94DA000C9400007A
:10002C000C941B010C9400000C9400000C94000028
:10003C000C9400000C9400000C9400000C94000034
:10004C000C9400000C9400000000F894EE27ECBB1C
:10005C00F1E0FBBFEBBFE5BFF8E1F1BDE1BD8DE029
:10006C00A2E0BB27ED938A95E9F780E094E0A0E647
:10007C00ED930197E9F7E4E5F0E085919591009710
:10008C0061F0A591B59105901590BF01F001059017
:10009C000D920197E1F7FB01F0CFEFE5EDBFE4E046
:1000AC00EEBFC0E6D1E00C943F01E5E0E3BF08955C
:1000BC00E0E0EFBDE9E0EEBD8D9A0895E881E3360E
:1000CC0051F421FE03C0E8EFF9E102C0E2E8FBE1E4
:1000DC00FBBDABD45C0E43651F421FE03C0E4E2A9
:1000EC00F7E102C0E3E8F8E1FBBDABD39C0E53653
:1000FC0019F4E6EDF5E132C0E63651F421FE03C009
:10010C00E5E7F3E102C0EDE9F4E1FBBDABD28C08F
:10011C00E73651F421FE03C0E5E5F1E102C0EDE55F
```

LAB1 (assembly projekt)



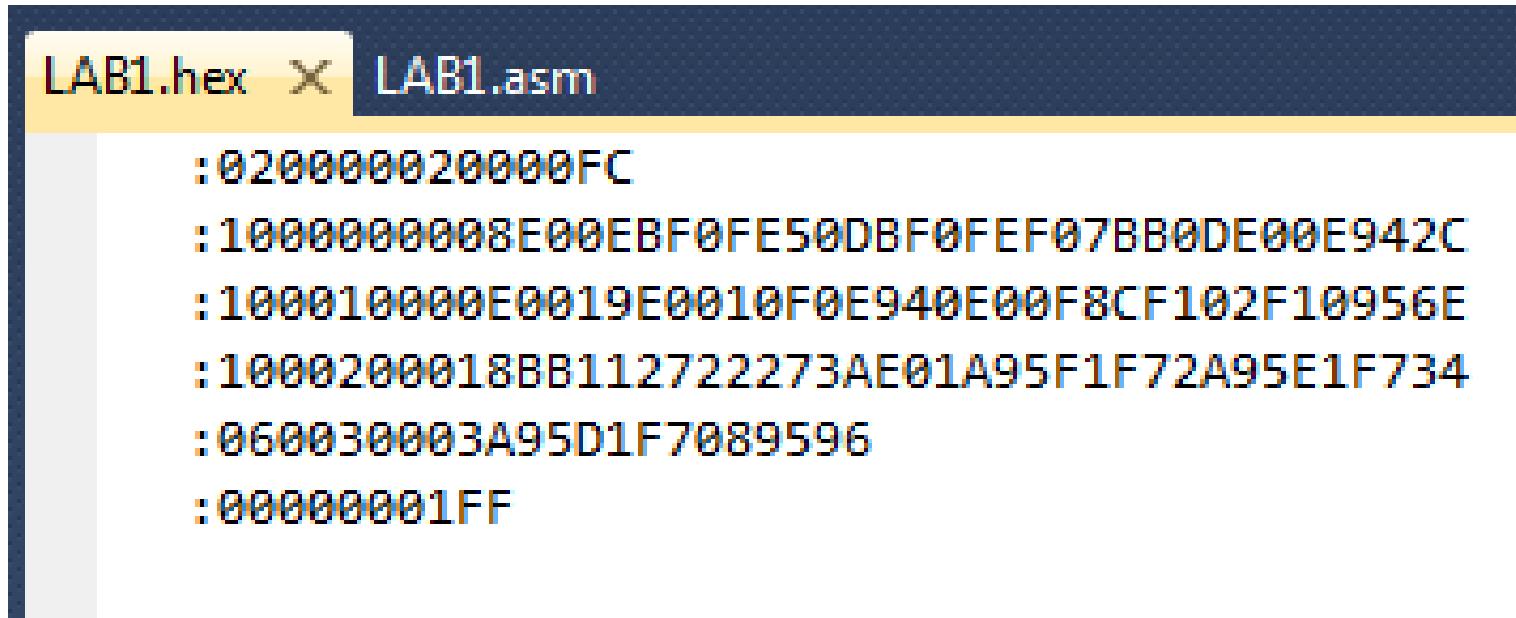
LAB1.asm

```
LAB1.asm X

;***** MSYS, LAB1 *****
;***** Henning Hargaard *****
;***** 14.august 2015 *****
;***** INITIERING *****
LDI R16,HIGH(RAMEND) ;Initialize Stack Pointer
OUT SPH,R16
LDI R16,LOW(RAMEND)
OUT SPL,R16
SER R16 ;PORTB = Outputs
OUT DDRB,R16

;***** PROGRAM-LOOP *****
CLR R16
LOOP:
LDI R17,9 ;R17 = 9
ADD R16,R17 ;R16 = R16 + R17
CALL DISP_AND_DELAY ;Display R16
JMP LOOP ;Jump to "LOOP"
```

LAB1.hex



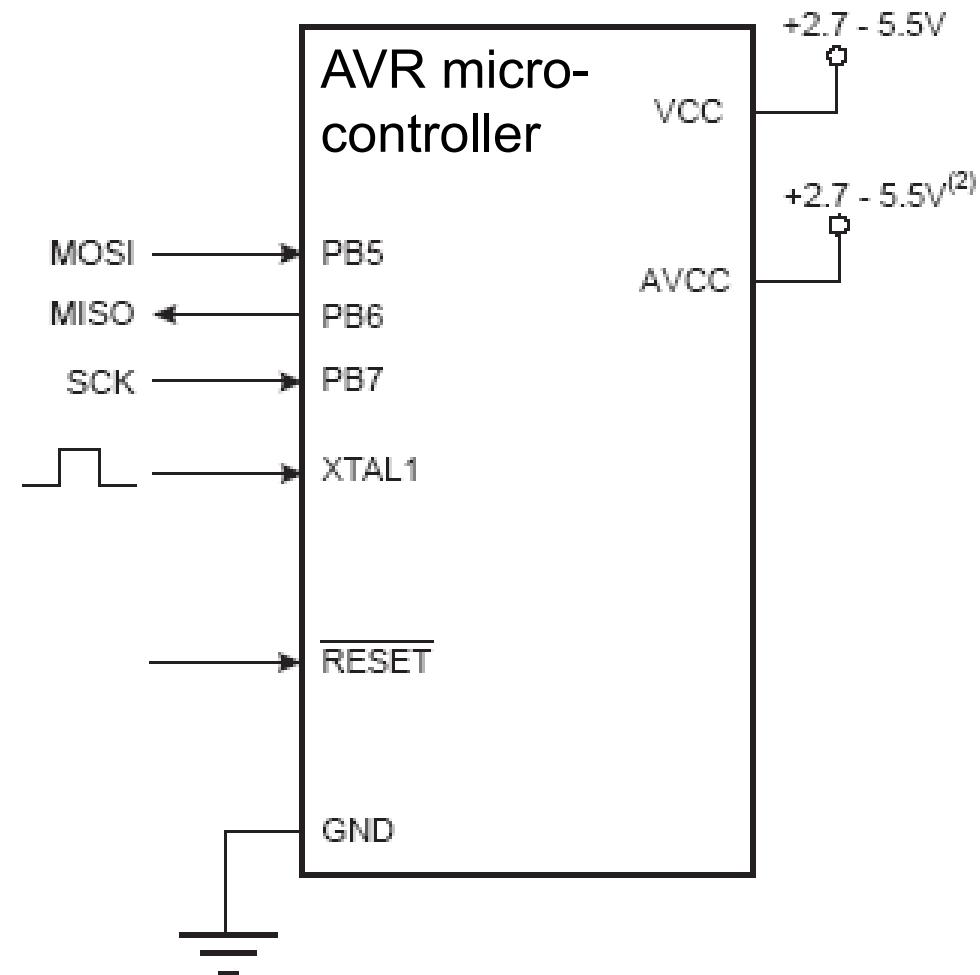
LAB1.hex X LAB1.asm

```
:020000020000FC
:1000000008E00EBF0FE50DBF0FEF07BB0DE00E942C
:100010000E0019E0010F0E940E00F8CF102F10956E
:1000200018BB112722273AE01A95F1F72A95E1F734
:060030003A95D1F7089596
:00000001FF
```

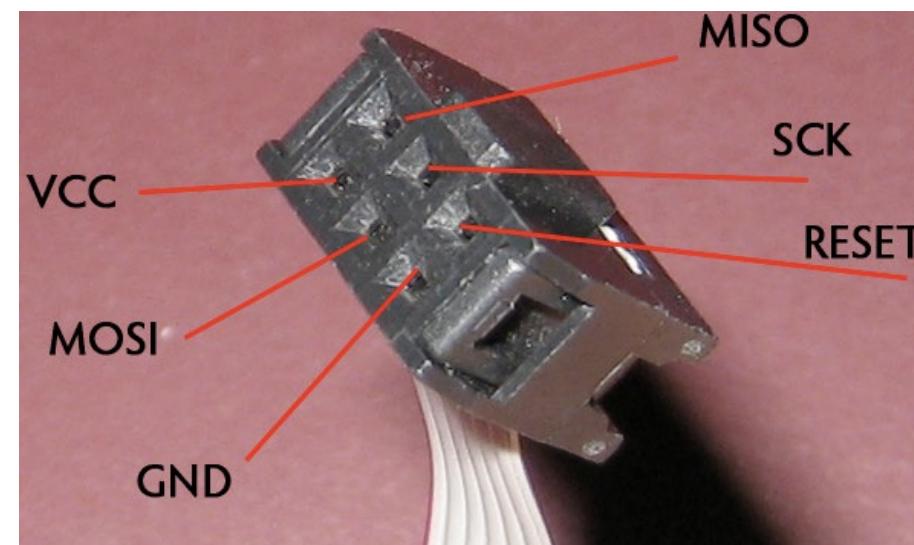
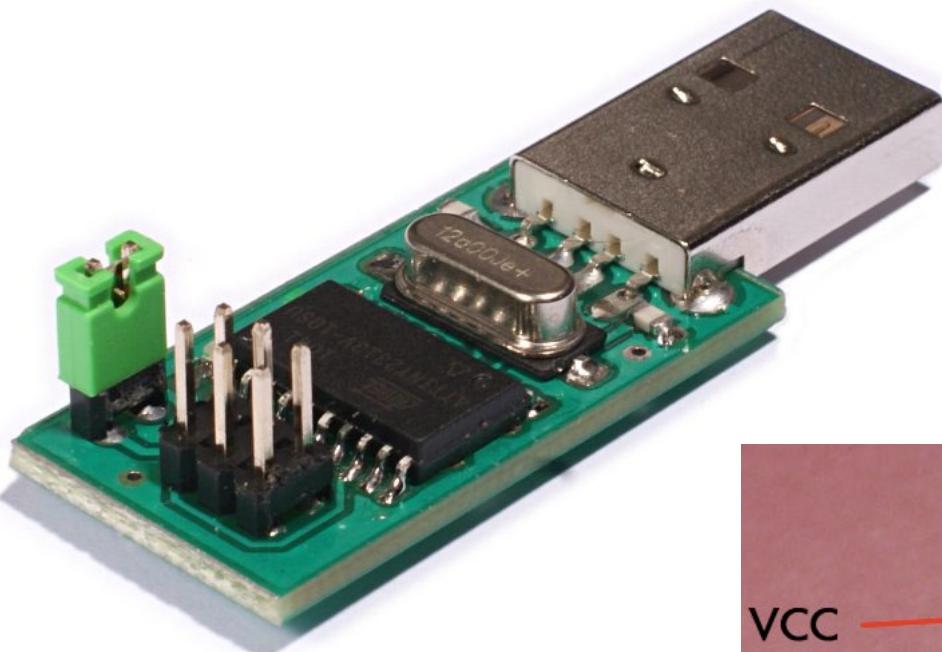
AVR programmerings-metoder

- Parallel programming (anvendes sjældent).
- **ISP** = In System Programming (anvendes meget ofte).
 - SPI
 - JTAG (kræver ekstra – ret dyrt – udstyr).
- **Bootloading** (speciel teknik, der kræver specialprogram på microcontrolleren).
“Arduino Mega2560” boardet har en bootloader, der er programmeret fra fabrikken.

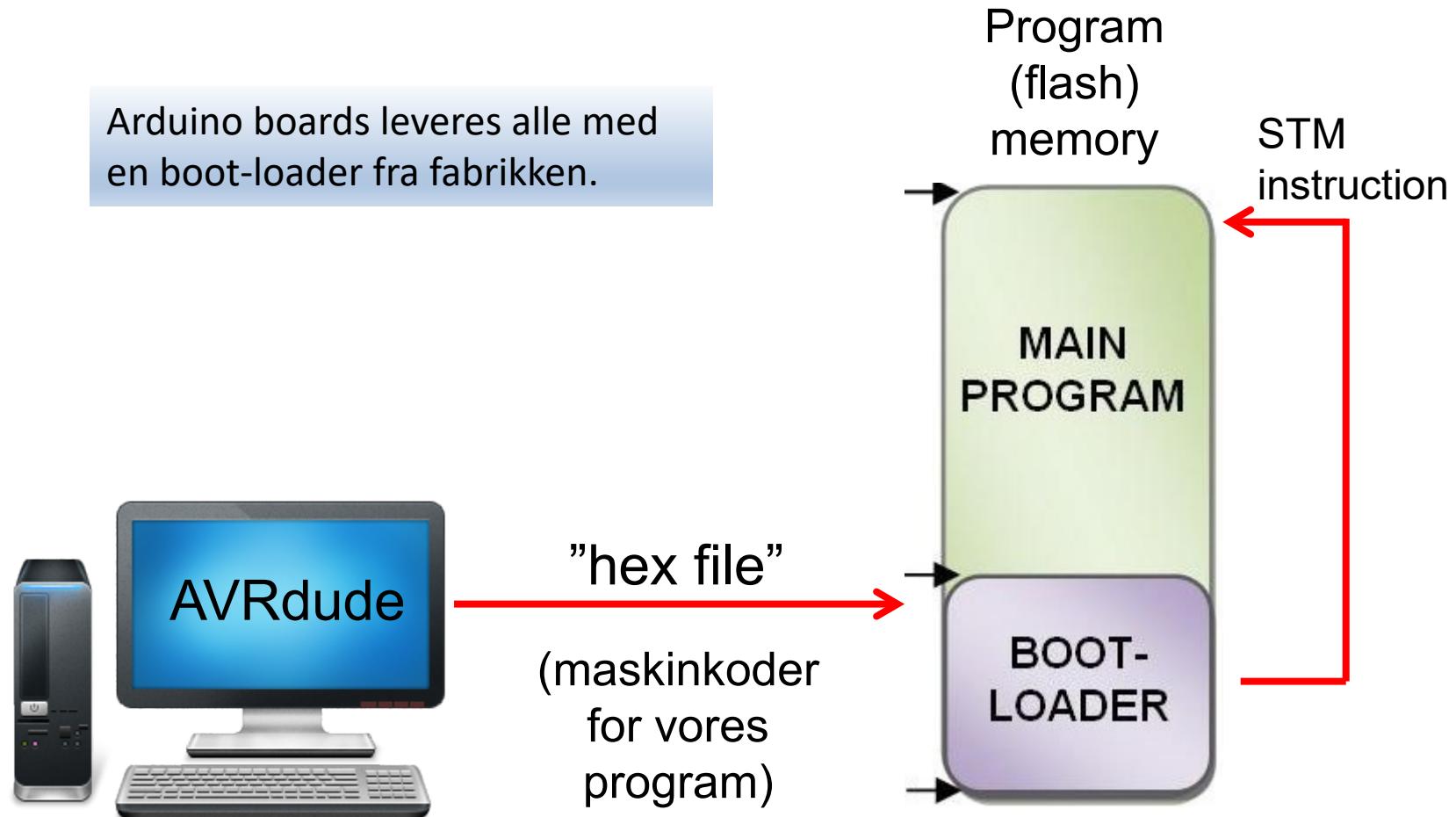
ISP seriell programmering



ISP programmer



Boot loading



Slut på lektion 11



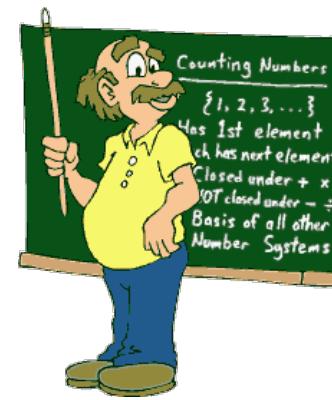
MSYS

Microcontroller Systems

Lektion 12 : Indkapsling

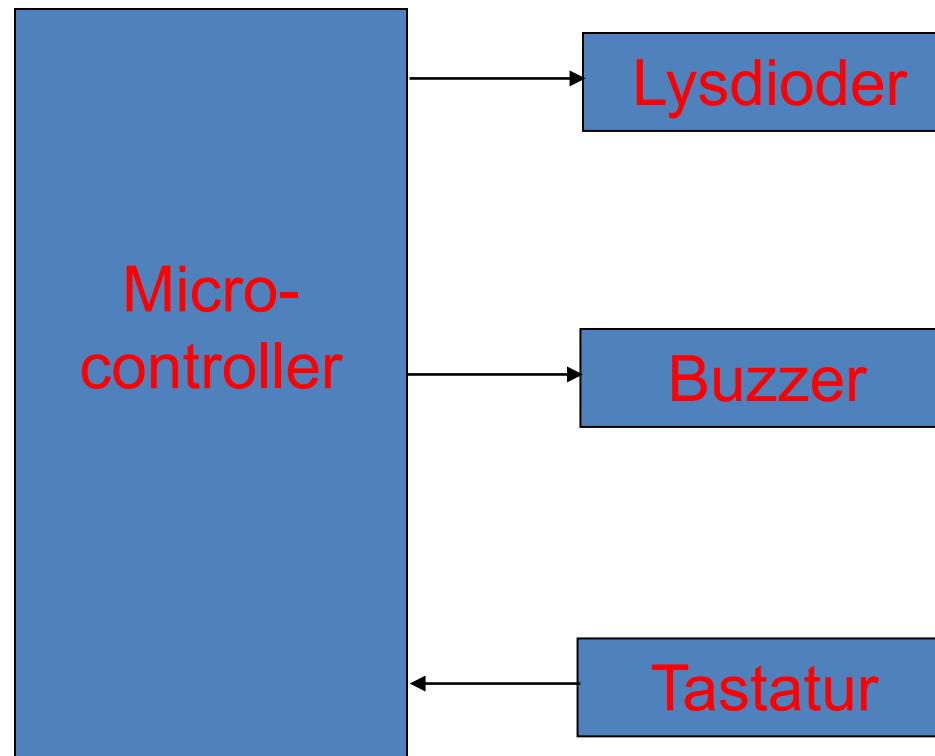


Slides from Henning Hargaard



Michael Alrøe

Design - eksempel



Ønsket funktionalitet (forenklet):

Ved tryk på taster på tastaturet, skal buzzeren afgive en tone i 1 sekund. Imens skal en bestemt lysdiode lyse.

Den ustrukturerede måde ("spaghetti")

```
void main()
{
    [Kode til klargøring af lysdioder]
    [Kode til klargøring af buzzer]
    [Kode til klargøring af tastatur]
    while(1)
    {
        if [Der er trykket på en tast]
        {
            [Start lydgiveren med den ønskede tone]
            [Tænd den ønskede lysdiode]
            [Vent 1 sekund]
            [Sluk lydgiveren]
            [Sluk alle lysdioder]
        }
    }
}
```

[...] er "pseudo"-kode



Problemet: Kode for forskellige H/W-enheder er i samme funktion og i samme fil. Rodet og uoverskuelig kode !

Bedre struktur: Opdeling i funktioner

```
void InitLysdioder()
{
    [Kode til klargøring af lysdioder]
}

void InitBuzzer()
{
    [Kode til klargøring af buzzer]
}

void InitTastatur()
{
    [Kode til klargøring af tastatur]
}

unsigned char TastTrykket()
{
    [Returner tastens kode, hvis der er trykket den]
    [- hvis der ikke nogen trykket: Returnerer 0]
}

void StartTone(unsigned char tast)
{
    [Kode, der starter tonen, som hører til "tast"]
}

void TaendLysdiode(unsigned char tast)
{
    [Kode, der tænder lysdioden, som hører til "tast"]
}

void StopTone()
{
    [Kode, der slukker lydgiveren]
}

void SlukAlleLysdioder()
{
    [Kode, der slukker alle lysdioder]
}
```

```
void main()
{
    unsigned char knap;

    InitLysdioder();
    InitBuzzer();
    InitTastatur();
    while(1)
    {
        knap = TastTrykket();
        if (knap)
        {
            StartTone(knap);
            TaendLysdiode(knap);
            [Vent 1 sekund]
            StopTone();
            SlukAlleLysdioder();
        }
    }
}
```

Gruppering efter HW

```
void InitLysdioder()
{
    [Kode til klargøring af lysdioder]
}

void InitBuzzer()
{
    [Kode til klargøring af buzzer]
}

void InitTastatur()
{
    [Kode til klargøring af tastatur]
}

unsigned char TastTrykket()
{
    [Returner tastens kode, hvis der er trykket den]
    [- hvis der ikke nogen trykket: Returnerer 0]
}

void StartTone(unsigned char tast)
{
    [Kode, der starter tonen, som hører til "tast"]
}

void TaendLysdiode(unsigned char tast)
{
    [Kode, der tænder lysdioden, som hører til "tast"]
}

void StopTone()
{
    [Kode, der slukker lydgiveren]
}

void SlukAlleLysdioder()
{
    [Kode, der slukker alle lysdioder]
}
```

```
void main()
{
    unsigned char knap;

    InitLysdioder();
    InitBuzzer();
    InitTastatur();
    while(1)
    {
        knap = TastTrykket();
        if (knap)
        {
            StartTone(knap);
            TaendLysdiode(knap);
            [Vent 1 sekund]
            StopTone();
            SlukAlleLysdioder();
        }
    }
}
```

— = Lysdioder

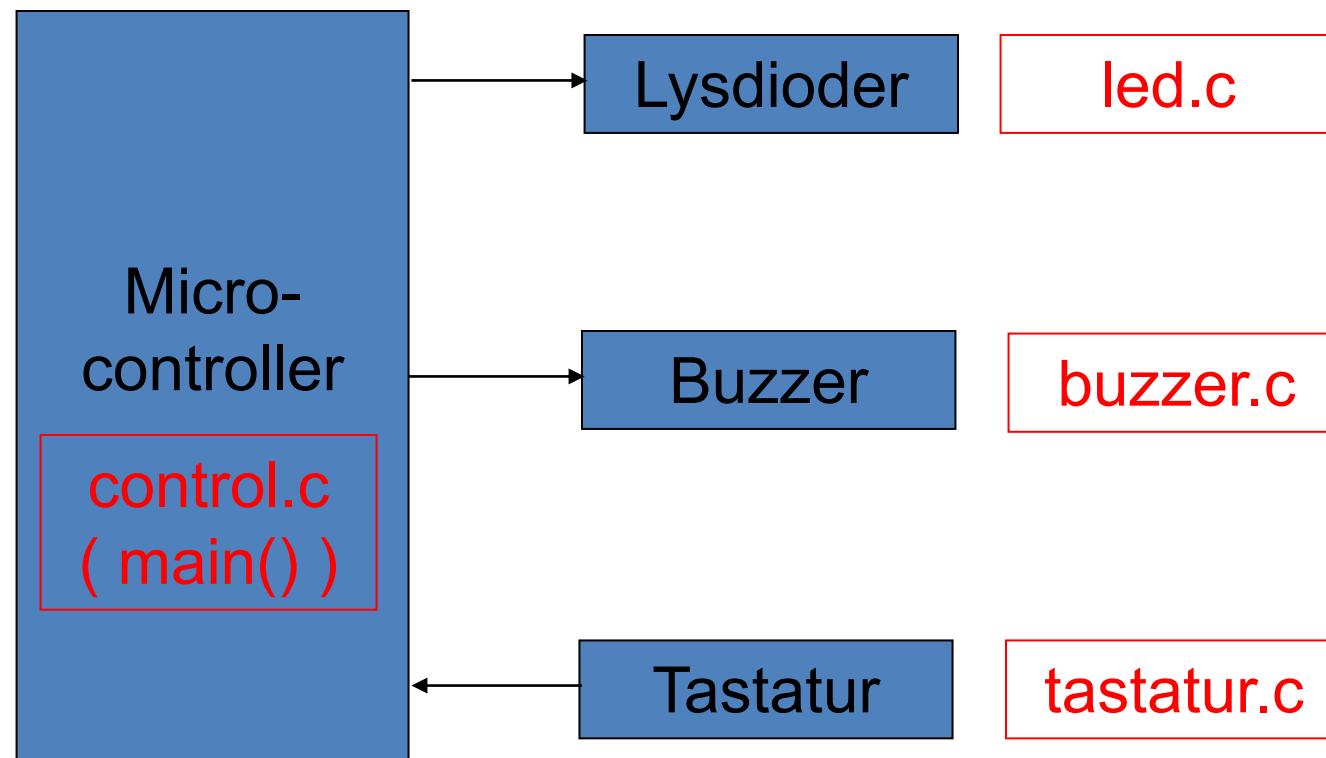
— = Buzzer

— = Tastatur

Modulering: En C-fil for hver HW-enhed

- Ved at skrive en C-fil for hver HW-enhed (gælder principielt også for SW-enheder), opdeler vi programmet i logiske blokke eller moduler.
- Bedre overskuelighed (= færre fejl).
- Bedre vedligeholdelses-venlighed.
- Mulighed for genbrug (en HW-enheds C-fil vil ofte kunne anvendes uforandret i et andet program). Skal derfor kun skrives (og testes) en gang for alle.
- Indkapsling (= beskyttelse) af data og funktioner.

Modul-opdeling (eksempel)



Lav os se på indholdet af
de 4 C-filer...

Filen "led.c"

```
void InitLysdioder()
{
    [Kode til klargøring af lysdioder]
}

void TaendLysdiode(unsigned char tast)
{
    [Kode, der tænder lysdioden, som hører til "tast"]
}

void SlukAlleLysdioder()
{
    [Kode, der slukker alle lysdioder]
}
```

Filen "buzzer.c"

```
void InitBuzzer()
{
    [Kode til klargøring af buzzer]
}

void StartTone(unsigned char tast)
{
    [Kode, der starter tonen, som hører til "tast"]
}

void StopTone()
{
    [Kode, der slukker lydgiveren]
}
```

Filen "tastatur.c"

```
void InitTastatur()
{
    [Kode til klargøring af tastatur]
}

unsigned char TastTrykket()
{
    [Returner tastens kode, hvis der er trykket den]
    [- hvis der ikke nogen trykket: Returnerer 0]
}
```

Filen "control.c"

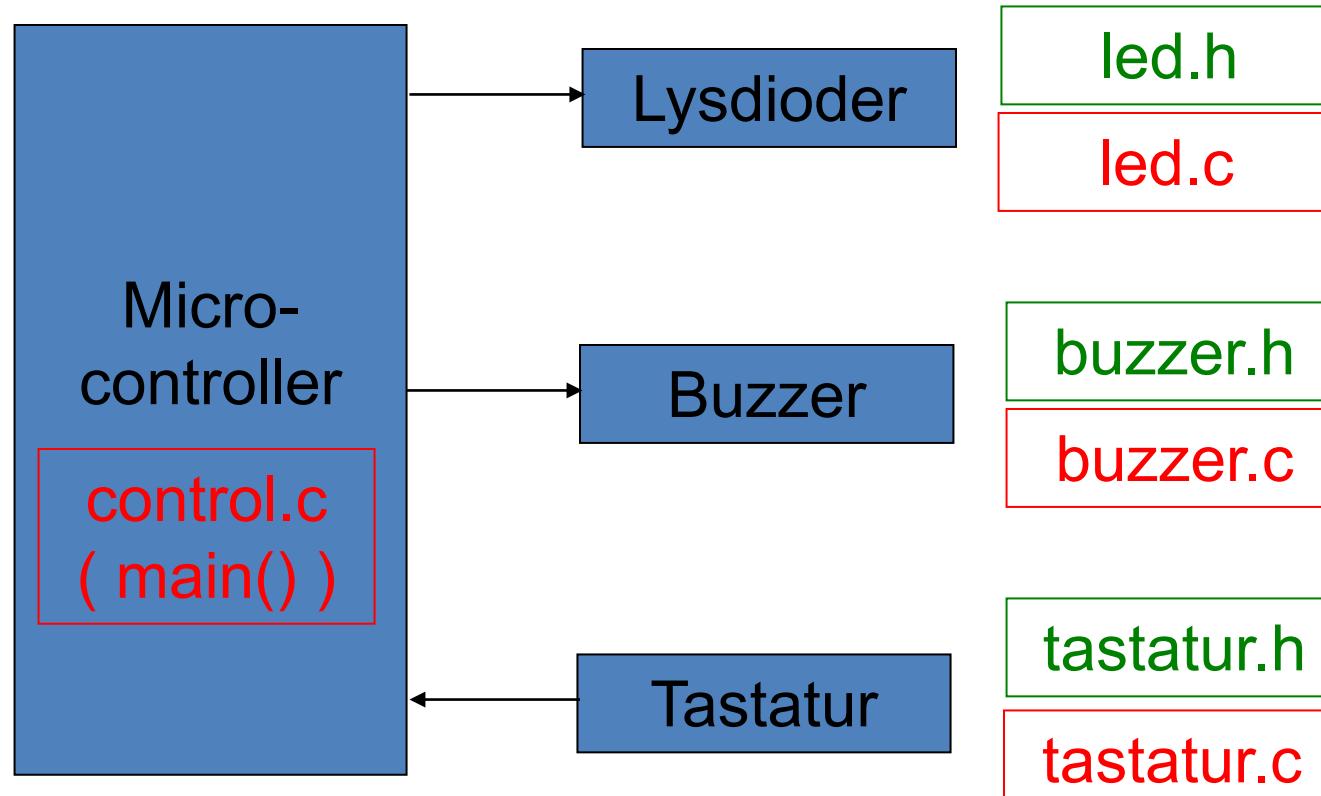
```
void main()
{
    unsigned char knap;

    InitLysdioder();
    InitBuzzer();
    InitTastatur();

    while(1)
    {
        knap = TastTrykket();
        if (knap)
        {
            StartTone(knap);
            TaendLysdiode(knap);
            [Vent 1 sekund]
            StopTone();
            SlukAlleLysdioder();
        }
    }
}
```

- De enkelte C-filer (her 4) **compileres en ad gangen**, hvorefter alt "linkes" til et program.
- **PROBLEM:** Når "control.c" oversættes, **kender compileren ikke prototypen** for de kaldte funktioner.
- Løsningen herpå er "**header-filer**" (.h -filer).

Header-filer (.h –filer)



Headerfilen indeholder prototyperne for de af modulets funktioner, der skal kunne kaldes fra andre moduler (her control.c).

Lav os se på indholdet af de 3 header-filer...

Indholdet af de 3 header-filer

"led.h":

```
// Prototyper for funktionerne i "led.c"
void InitLysdioder();
void TaendLysdiode(unsigned char tast);
void SlukAlleLysdioder();
```

"buzzer.h":

```
// Prototyper for funktionerne i "buzzer.c"
void InitBuzzer();
void StartTone(unsigned char tast);
void StopTone();
```

"tastatur.h":

```
// Prototyper for funktionerne i "tastatur.c"
void InitTastatur();
unsigned char TastTrykket();
```

Header-filerne
skal inkluderes
fra "control.c"....

Inkludering af headerfiler

```
#include "led.h"
#include "buzzer.h"
#include "tastatur.h"

void main()
{
    unsigned char knap;

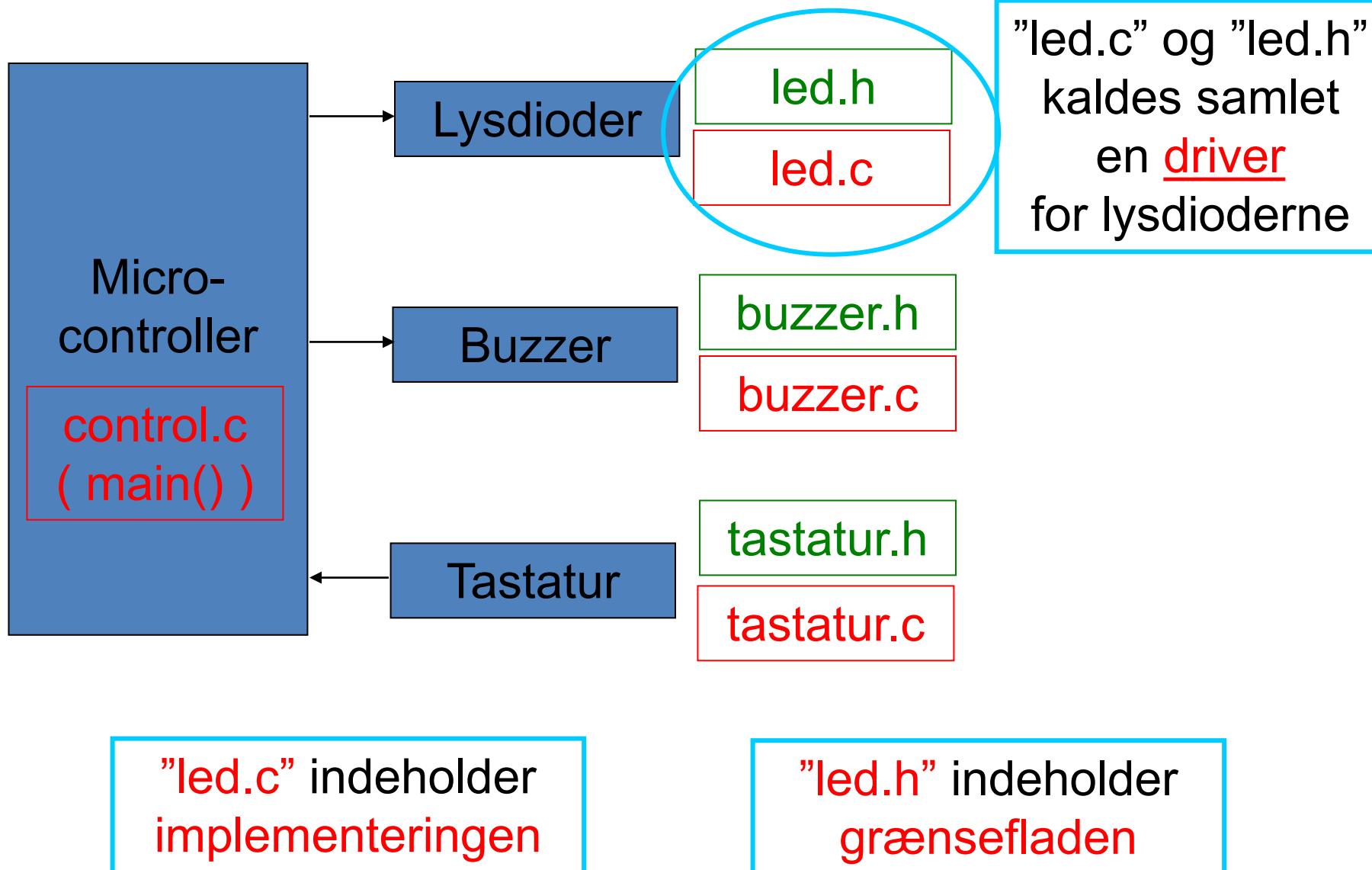
    InitLysdioder();
    InitBuzzer();
    InitTastatur();
    while(1)
    {
        knap = TastTrykket();
        if (knap)
        {
            StartTone(knap);
            TaendLysdiode(knap);
            [Vent 1 sekund]
            StopTone();
            SlukAlleLysdioder();
        }
    }
}
```

- **#include** af vores header-filer.
Compileren kan nu oversætte "control.c".

Problemet er løst !



Driver - begrebet



Øvelsen: LED driver

```
/* **** * **** * **** * **** * **** * **** *  
 * "LED.H" *  
 * Header file for "Mega2560 LED driver" *  
 * Henning Hargaard, 22/9 2015 *  
 * **** * **** * **** * **** * **** * **** */  
void init_LEDport();  
void writeAllLEDs(unsigned char pattern);  
void turnOnLED(unsigned char led_nr);  
void turnOffLED(unsigned char led_nr);  
void toggleLED(unsigned char led_nr);  
/* **** * **** * **** * **** * **** */
```

Øvelsen: Switch driver

```
*****  
* "Switch.h"                                *  
* Header file for "Mega2560 I/O Shield" SWITCH driver.      *  
* Henning Hargaard, 23/9 2015                  *  
*****/  
void initSwitchPort();  
unsigned char switchStatus();  
unsigned char switchOn(unsigned char switch_nr);  
*****/
```

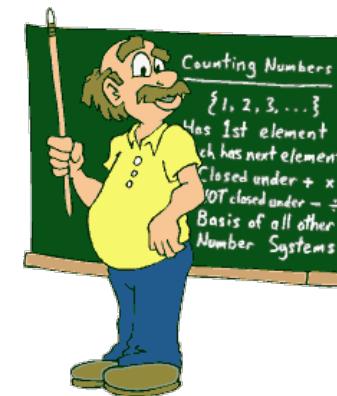
Slut på lektion 12



MSYS

Microcontroller Systems

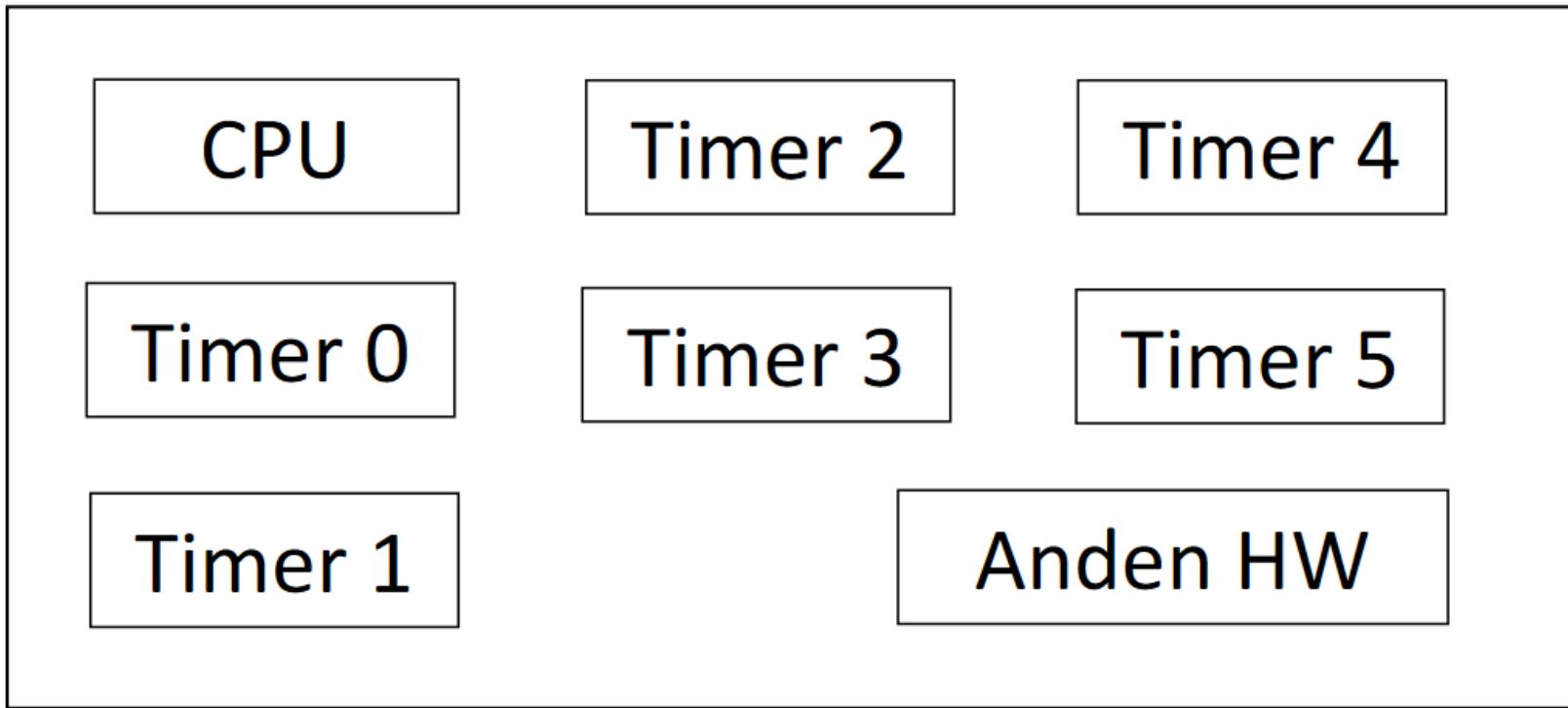
Lektion 13: Timers i Normal Mode



Slides from Henning Hargaard

Michael Alrøe

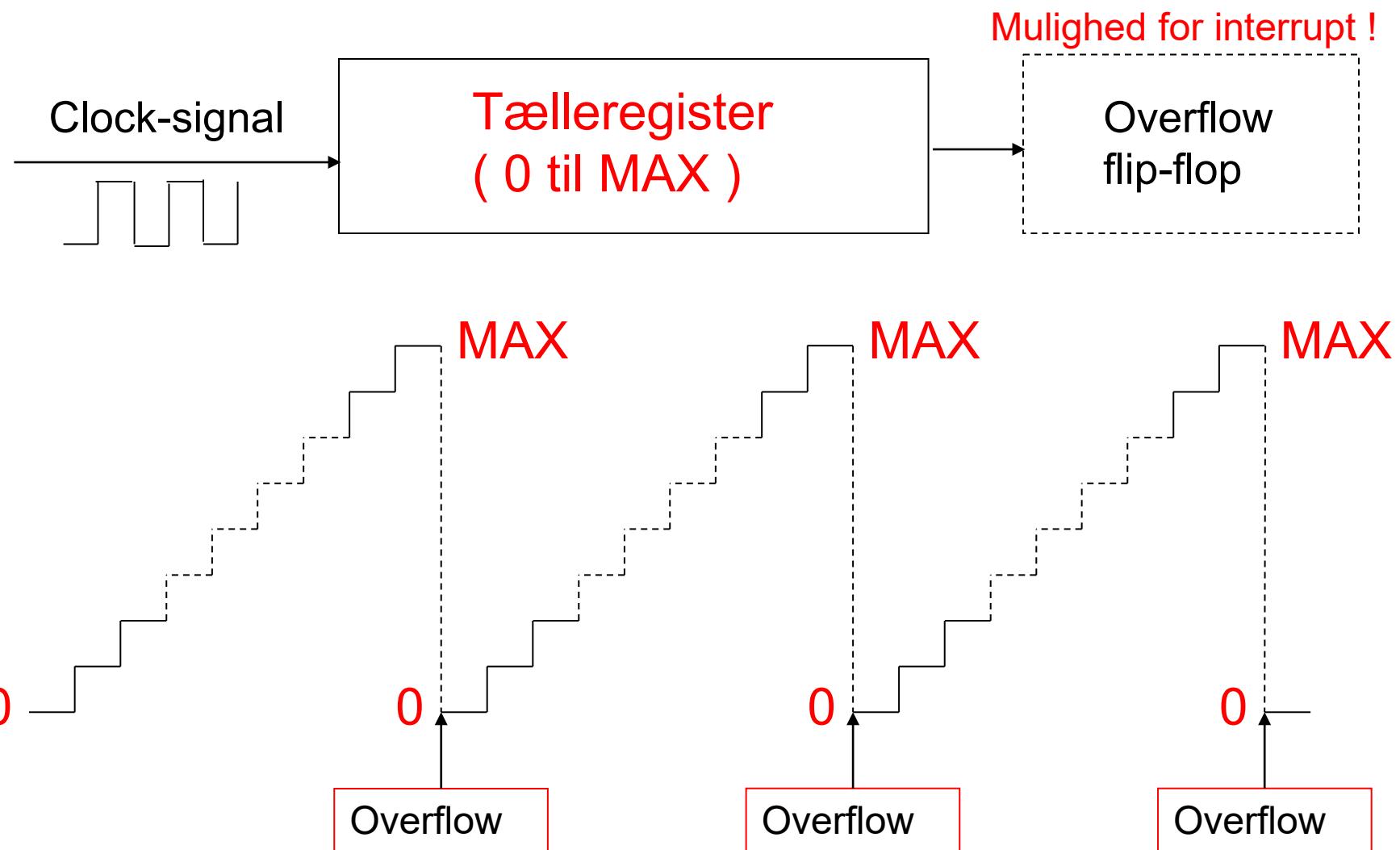
Timerne er selvstændige HW-enheder



**Timerne ”forstyrrer” altså ikke CPU’ens
programafvikling (eller omvendt).**

De ”arbejder” parallelt i tid.

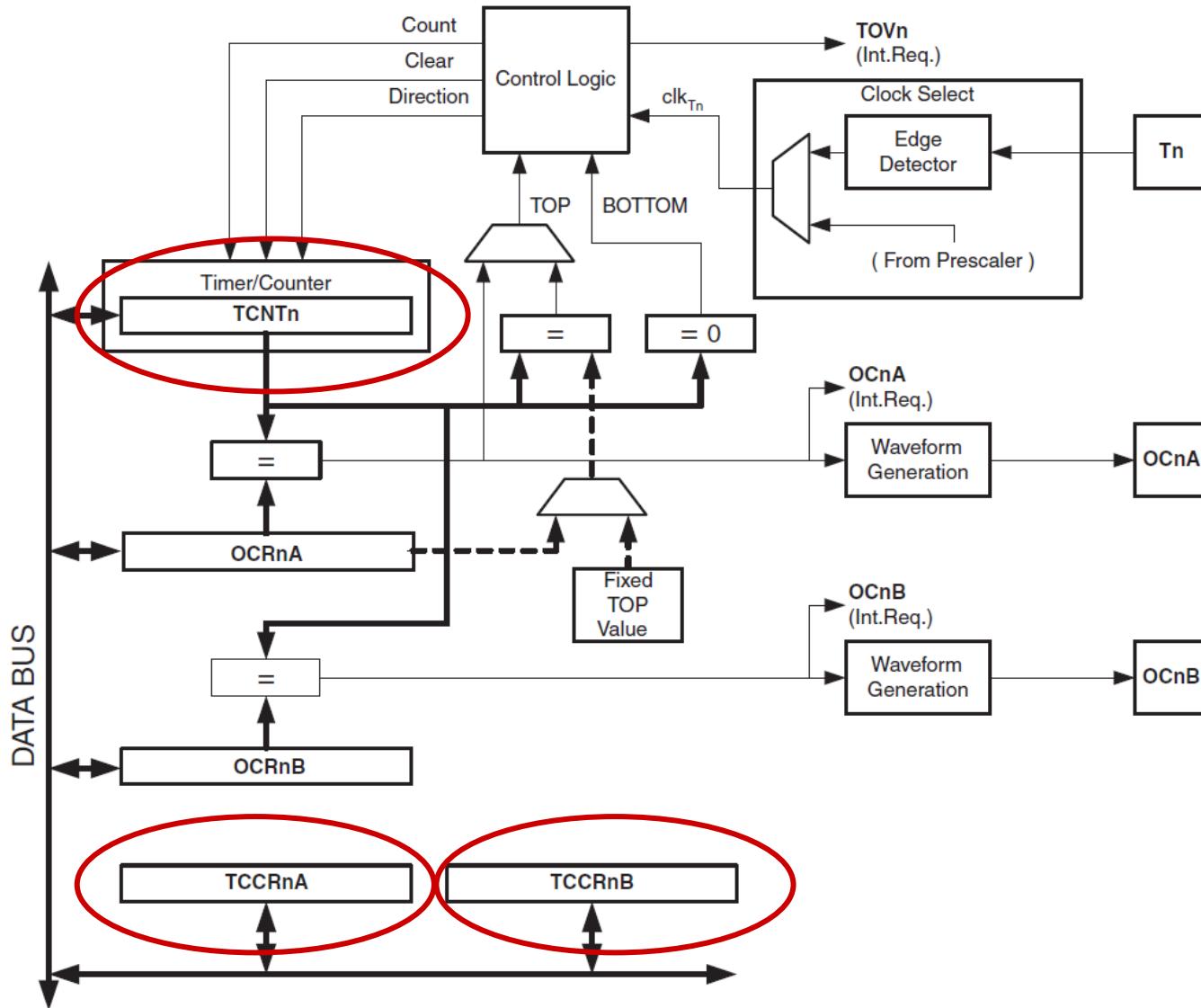
Timer i "normal mode"



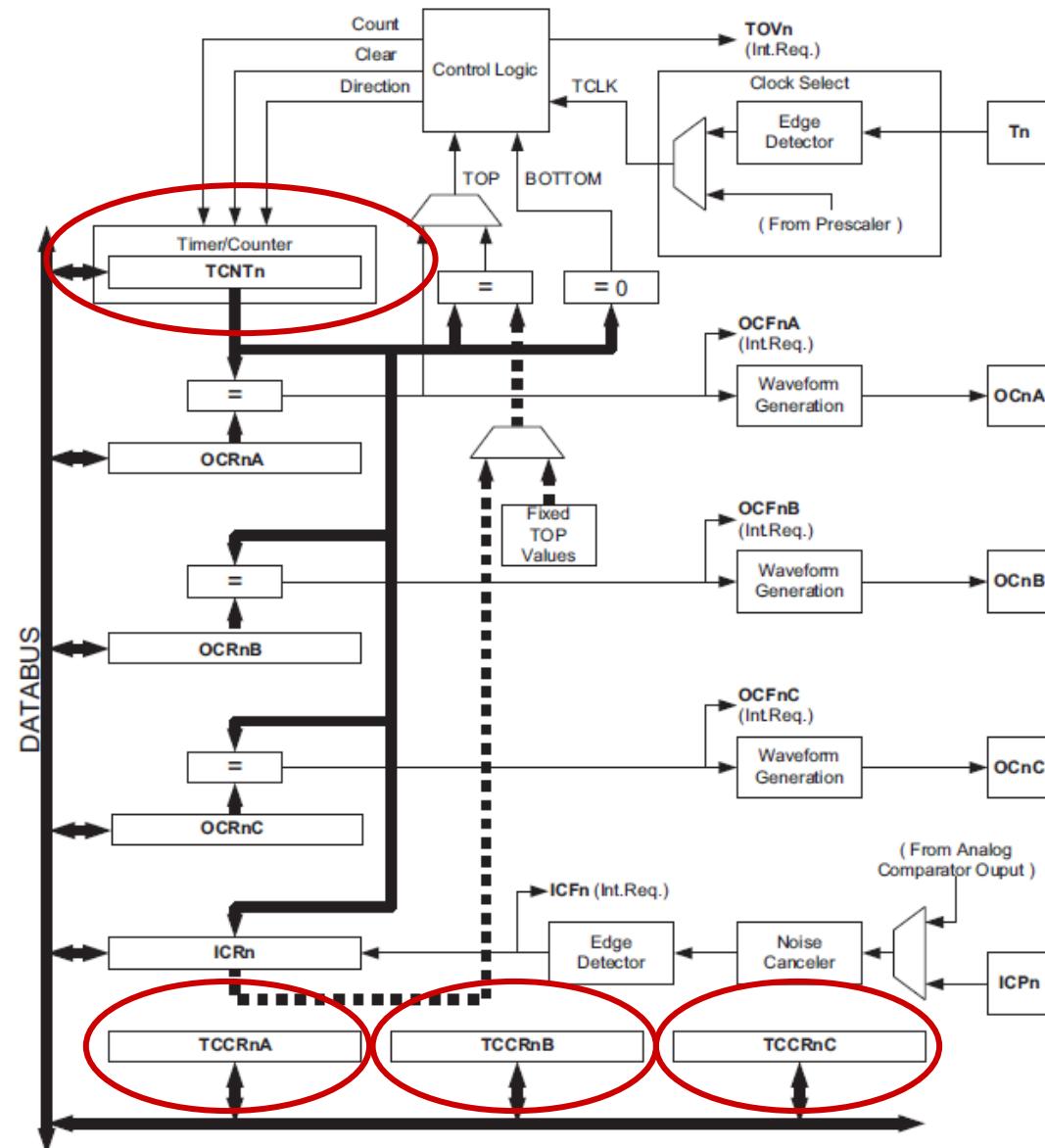
Mega2560: 6 timere

- **Timer 0 :**
8 bit (MAX = 255).
Normal, CTC og PWM modes.
- **Timer 1, Timer 3, Timer 4 og Timer 5 :**
16 bit (MAX = 65535).
Normal, CTC, mange PWM modes.
(Mulighed for "Input Capture")
- **Timer 2 :**
8 bit (MAX = 255).
Normal, CTC og PWM modes.
Asynkron mode (Real Time Clock).

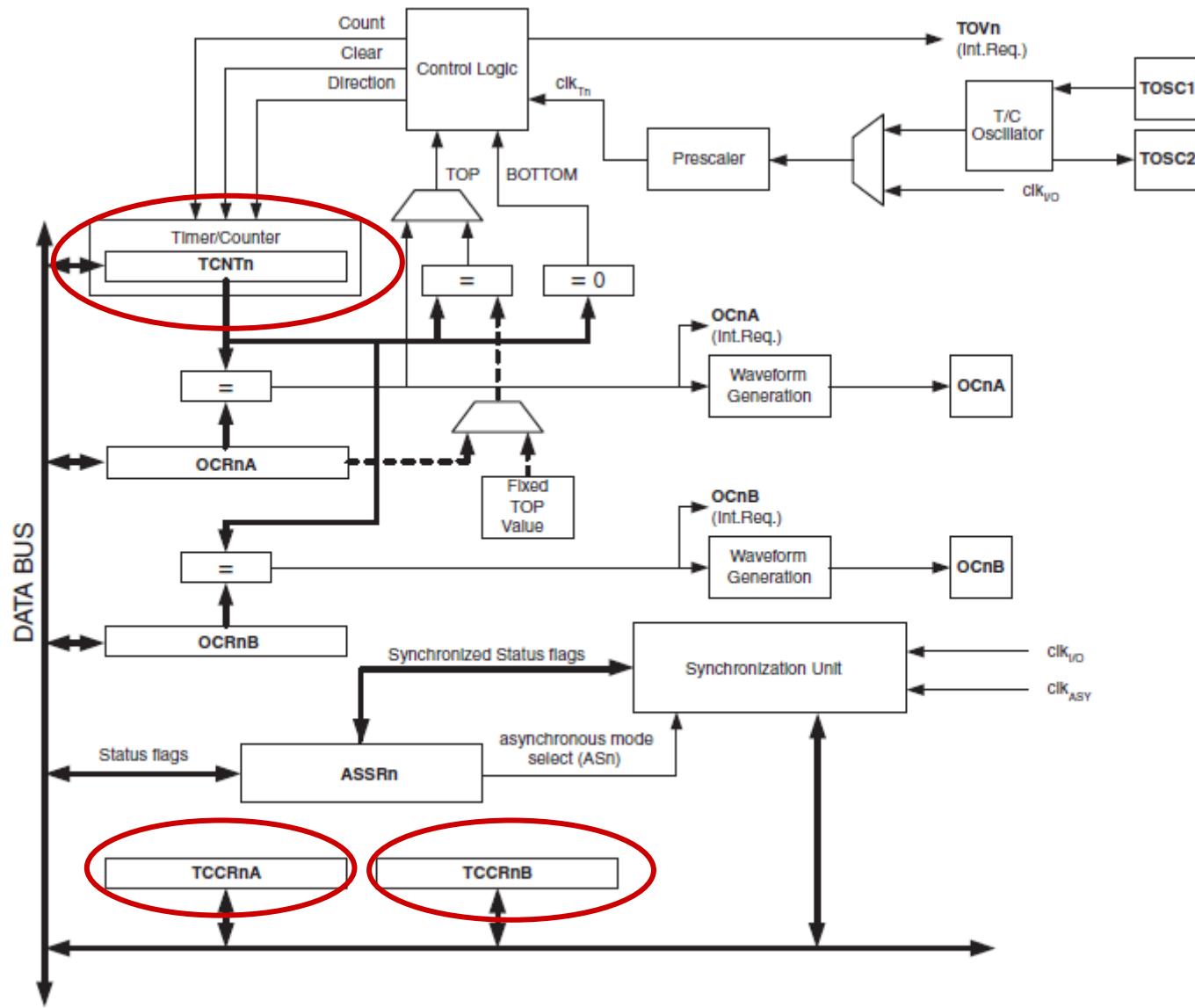
Mega2560: Timer 0 (8 bit)



Mega2560: Timer 1,3,4,5 (16 bit)



Mega2560: Timer 2 (8 bit)



Timer register (8 bit)

Bit	7	6	5	4	3	2	1	0	TCNT0
Read/Write	R/W								
Initial Value	0	0	0	0	0	0	0	0	

- **TCNTn** er selve tælle-registeret for timeren.
- **Optælles automatisk** af timer n clock-signalet.
- Bemærk, at registeret både kan læses fra og skrives til (fra programmet).

Timer register (16 bit)

Bit	7	6	5	4	3	2	1	0	
Read/Write	R/W	TCNT1H							
Initial Value	0	0	0	0	0	0	0	0	TCNT1L

- **TCNTn** er selve tælle-registeret for timer n.
- **Optælles automatisk** af timer n clock-signalet.
- Bemærk, at registeret både kan læses fra og skrives til (fra programmet).

Timer register (16 bit)

Bit	7	6	5	4	3	2	1	0	TCNT1H	TCNT1L
Read/Write	R/W									
Initial Value	0	0	0	0	0	0	0	0		

AVR GCC C:

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

// Herefter er 16-bit adgang muligt:
TCNT1 = 12345;
unsigned int x = TCNT3;
```

- Rækkefølge ved skrivning:
TCNT1H, derefter TCNT1L.
- Rækkefølge ved læsning:
TCNT1L, derefter TCNT1H.
- Kun aktuelt, når assembly anvendes.

Valg af Normal Mode

- Normal mode vælges normalt under opstart (initiering).
- Hvilke registre, der skal skrives til, afhænger af, om vi bruger Mega32 eller Mega2560. Desuden afhænger det af, hvilken timer, der drejer sig om.
- Normal mode er "default" efter RESET.

Mega2560: Timer 0, Normal mode

7	6	5	4	3	2	1	0	
COM0A1	COM0Ao	COM0B1	COM0B0	-	-	WGM01	WGM00	TCCR0A
R/W	R/W	R/W	R/W	R	R	R/W	R/W	
0	0	0	0	0	0	0	0	

7	6	5	4	3	2	1	0	
FOCoA	FOCoB	-	-	WGM02	CS02	CS01	CS00	TCCR0B
W	W	R	R	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	

Mode	WGM2	WGM1	WGM0	Timer/Counter Mode of Operation	TOP
0	0	0	0	Normal	0xFF

Mega2560: Timer 1,3,4,5. Normal mode

7	6	5	4	3	2	1	0
COM1A1	COM1A0	COM1B1	COM1B0	COM1C1	COM1C0	WGM11	WGM10
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
7	6	5	4	3	2	1	0
ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10
R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

- TCCRnA = TCCR1A, TCCR3A, TCCR4A eller TCCR5A.
- TCCRnB = TCCR1B, TCCR3B, TCCR4B eller TCCR5B.

Mode	WGMr3	WGMr2 (CTCrn)	WGMr1 (PWMr1)	WGMr0 (PWMr0)	Timer/Counter Mode of Operation	TOP
0	0	0	0	0	Normal	0xFFFF

Mega2560: Timer 2, Normal mode

7	6	5	4	3	2	1	0	
COM2A1	COM2A0	COM2B1	COM2B0	-	-	WGM21	WGM20	TCCR2A
R/W	R/W	R/W	R/W	R	R	R/W	R/W	

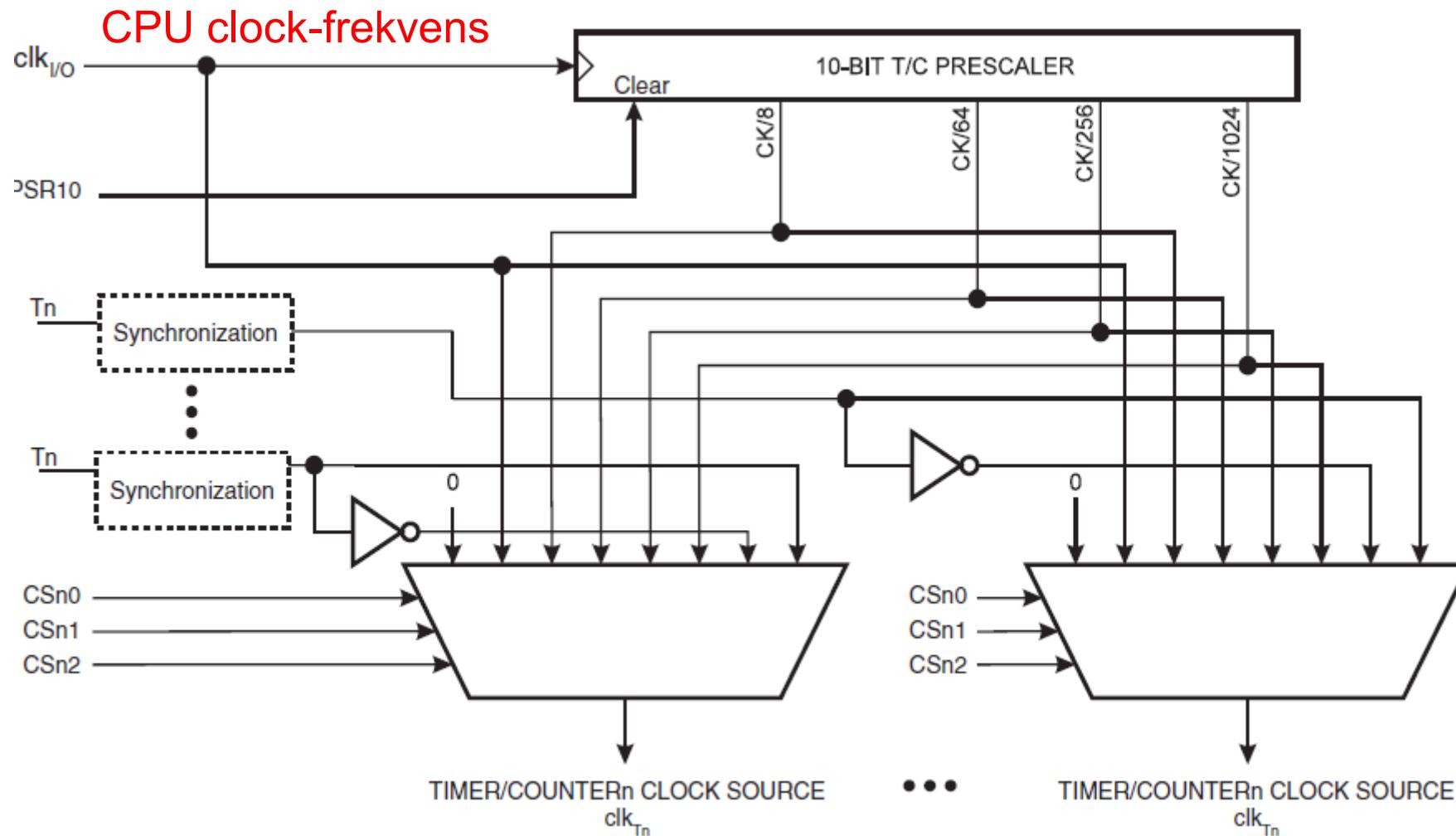
7	6	5	4	3	2	1	0	
FOC2A	FOC2B	-	-	WGM22	CS22	CS21	CS20	TCCR2B
W	W	R	R	R/W	R/W	R/W	R/W	

Mode	WGM2	WGM1	WGM0	Timer/Counter Mode of Operation	TOP
				Normal	
0	0	0	0		0xFF

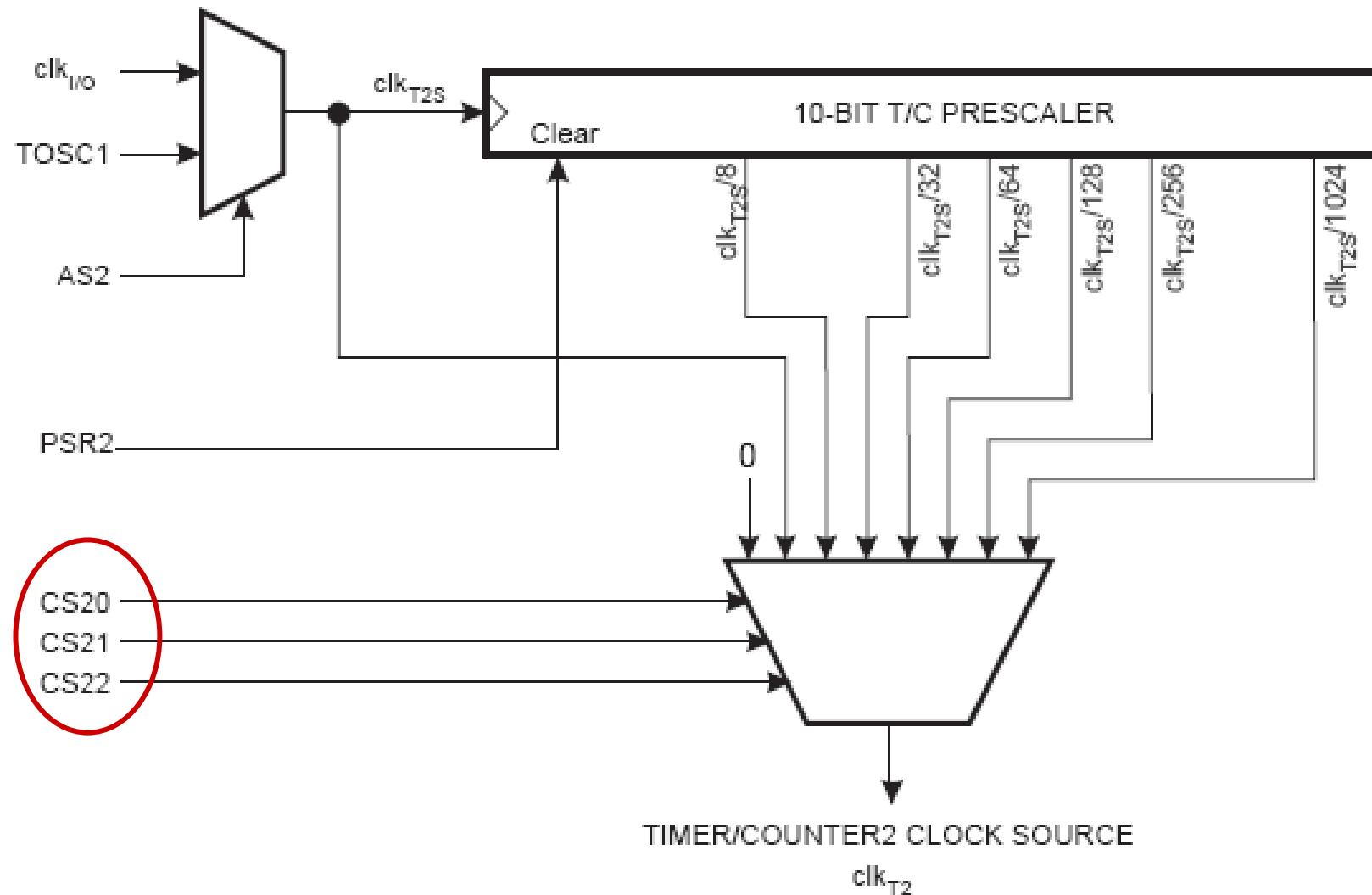
Valg af clocksignal (prescaling)

- En timers clocksignal vælges ved at indstille en "prescaler". Hver timer har en clock prescaler.
- Hvilke registre, der skal skrives til, afhænger af, om vi bruger Mega32 eller Mega2560. Desuden afhænger det af, hvilken timer, der drejer sig om.
- "No clock" er default efter RESET.

Timer Prescaler (ikke Timer 2)



Prescaler for timer 2



Mega2560: Timer 0. Valg af clock

7	6	5	4	3	2	1	0
FOC0A	FOC0B	-	-	WGM02	CS02	CS01	CS00
W	W	R	R	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$\text{clk}_{\text{I/O}}$ /(No prescaling)
0	1	0	$\text{clk}_{\text{I/O}}/8$ (From prescaler)
0	1	1	$\text{clk}_{\text{I/O}}/64$ (From prescaler)
1	0	0	$\text{clk}_{\text{I/O}}/256$ (From prescaler)
1	0	1	$\text{clk}_{\text{I/O}}/1024$ (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

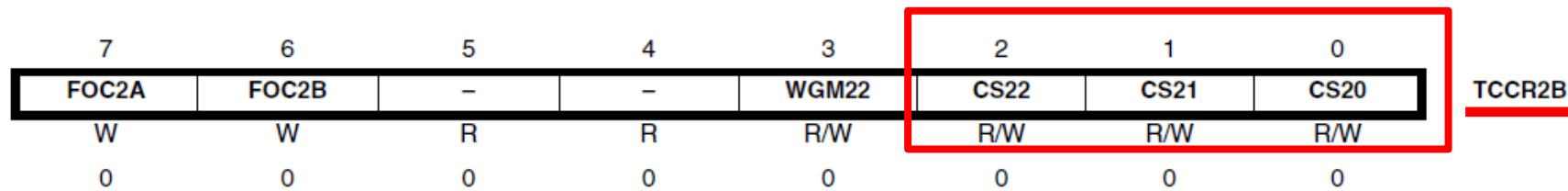
Mega2560: Timer 1,3,4,5. Valg af clock

7	6	5	4	3	2	1	0	
ICNC1	ICES1	-	WGM13	WGM12	CSn2	CSn1	CSn0	TCCRnB
R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	

- TCCRnB = TCCR1B, TCCR3B, TCCR4B eller TCCR5B.

CSn2	CSn1	CSn0	Description
0	0	0	No clock source. (Timer/Counter stopped)
0	0	1	$\text{clk}_{\text{I/O}}/1$ (No prescaling)
0	1	0	$\text{clk}_{\text{I/O}}/8$ (From prescaler)
0	1	1	$\text{clk}_{\text{I/O}}/64$ (From prescaler)
1	0	0	$\text{clk}_{\text{I/O}}/256$ (From prescaler)
1	0	1	$\text{clk}_{\text{I/O}}/1024$ (From prescaler)
1	1	0	External clock source on Tn pin. Clock on falling edge
1	1	1	External clock source on Tn pin. Clock on rising edge

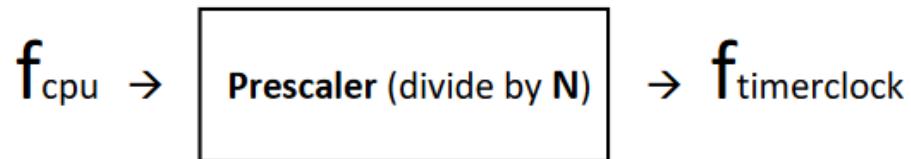
Mega2560: Timer 2, Valg af clock



CS22	CS21	CS20	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	clk_{T2S} /(No prescaling)
0	1	0	$\text{clk}_{\text{T2S}}/8$ (From prescaler)
0	1	1	$\text{clk}_{\text{T2S}}/32$ (From prescaler)
1	0	0	$\text{clk}_{\text{T2S}}/64$ (From prescaler)
1	0	1	$\text{clk}_{\text{T2S}}/128$ (From prescaler)
1	1	0	$\text{clk}_{\text{T2S}}/256$ (From prescaler)
1	1	1	$\text{clk}_{\text{T2S}}/1024$ (From prescaler)

Timer clocksignal og prescaler

Hver timer har sin egen prescaler, der kan neddele CPU clockfrekvensen med et heltal (kaldes N).



Altså: $f_{timerclock} = f_{cpu} / N$

Test ("socrative.com"): Room = AUQUIZ

- Antag, at vi anvender en CPU clockfrekvens på 3,6864 MHz, og at Timer 0 er initieret til Normal Mode.
Timer 0's clock prescaler er sat til 64.
Hvor ofte laver Timer 0 overflow ?
- A: 225 gange per sekund.
- B: 256 gange per sekund.
- C: 57600 gange per sekund.
- D: 64 gange per sekund.



Test ("socrative.com"): Room = AUQUIZ

- Antag Timer 1 er i Normal Mode og CPU clockfrekvensen er 16 MHz.
Hvad er den længste tid, der kan opnås mellem hvert Timer 1 overflow ?
- A: Ca. 244 sekunder.
- B: 15625 sekunder.
- C: Ca. 240 ms.
- D: Ca. 4,2 sekunder.



Eksterne clock-ben

- Hvis der for en timer er valgt "External Clock", tæller timeren pulser på et microcontrollerben.
- Hver timer (undtaget Timer 2) har et ben, der kan bruges som eksternt clocksignal.

Se næste slide.

Mega2560: Eksterne clock-ben

- Timer 0 :
T0 = Port D, ben 7
- Timer 1 :
T1 = Port D, ben 6
- Timer 3 :
T3 = Port E, ben 6
- Timer 4 :
T4 = Port H, ben 7
- Timer 5 :
T5 = Port L, ben 2



På "vores hardware":

T0 er lav (0), når vi
trykker på SW0.

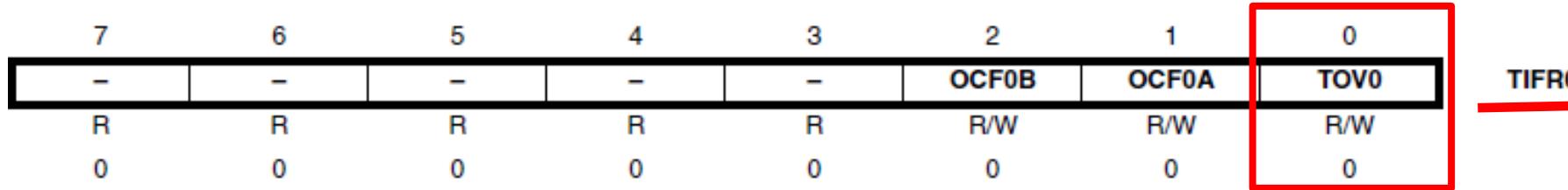
T0 er høj, når vi ikke
trykker på SW0.

Timer overflow flag

- Hver gang en timer laver overflow, vil den sætte et "overflow flag" (til 1). Flaget er et bit i et bestemt I/O register.
- Bliver kun nulstillet igen, hvis vi skriver et 1-tal til det (medmindre vi bruger interrupts). Det svarer til at "sænke flaget".
- Hvilke registre, der skal skrives til, afhænger af, om vi bruger Mega32 eller Mega2560. Desuden afhænger det af, hvilken timer, der drejer sig om.

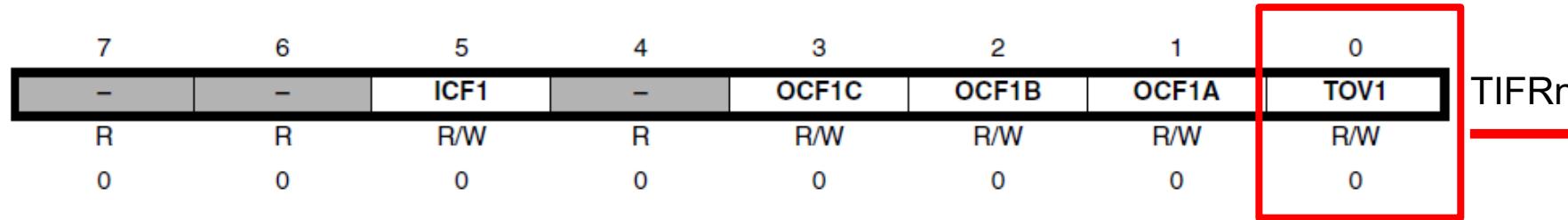


Mega2560: Timer 0 overflow flag



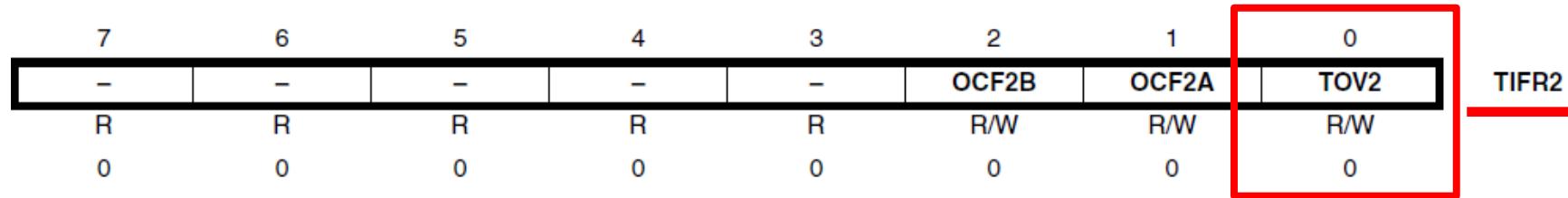
- Bit 0 i registeret TIFR0 bliver **sat til 1**, hver gang Timer 0 laver overflow.
- Kan kun nulstilles igen, hvis vi skriver et 1-tal til det (medmindre vi bruger interrupts).
- C-kode:
TIFR0 = 0b00000001;

Mega2560: Timer 1,3,4,5. Overflow flags



- **TIFR n** = TIFR1, TIFR3, TIFR4 eller TIFR5.
- Bit 0 i registeret TIFR" n " bliver **sat til 1**, hver gang Timer " n " laver **overflow**.
- Kan kun nulstilles igen, hvis vi skriver et 1-tal til det (medmindre vi bruger interrupts).
- C-kode:
TIFR4 = 0b00000001;

Mega2560: Timer 2 overflow flag



- Bit 0 i registeret TIFR2 bliver **sat til 1**, hver gang Timer 2 laver overflow.
- Kan kun nulstilles igen, hvis vi skriver et 1-tal til det (medmindre vi bruger interrupts).
- C-kode:
TIFR0 = 0b00000001;

Delay med Timer i Normal mode

Metode:

1. Skriv en beregnet værdi **x** til tælleregisteret (TCNTn).
2. Giv timeren clocksignal (skriv til **prescaler**-bits).
3. Vent på, at overflow flaget bliver 1.
4. Fjern timerens clocksignal (vælg "no clock").
5. Nulstil overflow flaget (ved at skrive 1 til det).

Tidsforsinkelsens størrelse bestemmes af punkt 1 og 2:

Timerens clockfrekvens f_{timer}: f_{CPU} / N (prescaler)

Tid for 1 trin opad "trappen": $T_{timer} = 1 / f_{timer} = N / f_{CPU}$

Samlet delay: $T_{delay} = \text{"Antal trin til overflow"} * T_{timer}$

$$T_{delay} [\text{sekunder}] = ((\text{"Max for timeren + 1"} (\text{256 eller 65536}) - x) * N) / f_{CPU}$$

Eksempel fra bogen: Timer 0

Example 9-39

Write a C program to toggle all the bits of PORTB continuously with some delay. Use Timer0, normal mode, and no prescaler options to generate the delay.

Solution:

```
#include "avr/io.h"
void T0Delay ( );
int main ( )
{
    DDRB = 0xFF;

    while (1)
    {
        PORTB =
        T0Delay
        PORTB =
        T0Delay
    }
}

void T0Delay ( )
{
    TCNT0 = 0x20;           //load TCNT0
    TCCR0 = 0x01;           //Timer0, Normal mode, no prescaler
    while ((TIFR&0x1)==0); //wait for TF0 to roll over
    TCCR0 = 0;
    TIFR = 0x1;             //clear TF0
}
```



Mega2560: Timer 0 overflow 10000 gange i sekundet

```
void T0Delay()
{
    // 16000000 Hz /8 = 2000000 Hz
    // 2000000 Hz / 200 = 10000 Hz
    TCNT0 = 256-200;
    // Timer 0 i Normal mode og PS = 8
    TCCR0A = 0b00000000;
    TCCR0B = 0b00000010;
    // Vent på Timer 0 overflow flag
    while (((TIFR0 & (1<<0)) == 0)
    {}
    // Stop Timer 0 (ingen clock)
    TCCR0B = 0b00000000;
    // Nulstil Timer 0 overflow flag
    TIFR0 = 0b00000001;
}
```



Eksempel fra bogen: Timer 1

Example 9-42 (C version of Example 9-32)

Write a C program to toggle only the PORTB.4 bit continuously every second. Use Timer1, Normal mode, and 1:256 prescaler to create the delay. Assume XTAL = 8 MHz.

Solution:

XTAL = 8 MHz \rightarrow $T_{\text{machine cycle}} = 1/8 \text{ MHz} = 0.125 \mu\text{s} = T_{\text{clock}}$

Prescaler = 1:256 \rightarrow $T_{\text{clock}} = 256 \times 0.125 \mu\text{s} = 32 \mu\text{s}$

$1 \text{ s}/32 \mu\text{s} = 31,250 \text{ clocks} = 0x7A12 \text{ clocks} \rightarrow 1 + 0xFFFF - 0x7A12 = 0x85EE$

```
#include "avr/io.h"

void T1Delay ( );

int main ( )
{
    DDRB = 0xFF;

    while (1)
    {
        PORTB = 0x00;
        T1Delay();
    }
}

void T1Delay ( )
{
    TCNT1H = 0x85;      //TEMP = 0x85
    TCNT1L = 0xEE;

    TCCR1A = 0x00;      //Normal mode
    TCCR1B = 0x04;      //Normal mode, 1:256 prescaler

    while ((TIFR&(0x1<<TOV1))==0);      //wait for TF0 to roll over

    TCCR1B = 0;
    TIFR = 0x1<<TOV1;      //clear TOV1
}
```



Mega2560: Timer 1 overflow hver 3. sekund

```
void T1Delay()
{
    // 16000000 Hz /1024 = 15625 Hz
    // Vi har altså 15625 "trin" per sekund
    // - og ønsker 3 sekunder til overflow (= 1/3 Hz)
    TCNT1 = 65536-(3*15625);
    // Timer 1 i Normal Mode og PS = 1024
    TCCR1A = 0b00000000;
    TCCR1B = 0b00000101;
    // Afvent Timer 1 overflow flag
    while ((TIFR1 & (1<<0)) == 0)
    {}
    // Stop Timer 1 (ingen clock)
    TCCR1B = 0b00000000;
    // Nulstil Timer 1 overflow flag
    TIFR1 = 1<<0;
}
```



Test ("socrative.com"): Room = AUQUIZ

```
void T0Delay()
{
    TCNT0 = 117;
    TCCR0A = 0b00000000;
    TCCR0B = 0b00000011;
    // Vent på Timer 0 overflow flag
    while ((TIFR0 & (1<<0)) == 0)
    {}
    // Stop Timer 0 (ingen clock)
    TCCR0B = 0b00000000;
    // Nulstil Timer 0 overflow flag
    TIFR0 = 0b00000001;
}
```

Mega2560 anvendes.
CPU frekvens = 16 MHz.
Hvad er forsinkelsen ?

A: 46,8 us

B: 8,7 us

C: 7,48 ms

D: 556 us

E: 7,31 us



Dagens LAB-øvelse

Øvelsen

I denne øvelse vil vi:

- Anvende Timer 0 i normal mode.
- Anvende Timer 1 i normal mode (et "binært ur" implementeres).
- Anvende Timer 0 til at tælle eksterne pulser.

Den i LAB7 udviklede LED driver vil blive genbrugt i denne øvelse.

Del 2: Det binære ur:

Lysdioderne 7 - 4	Lysdioderne 3 - 0
Sekund 10'ere (0-5)	Sekund 1'ere (0-9)

Slut på lektion 13

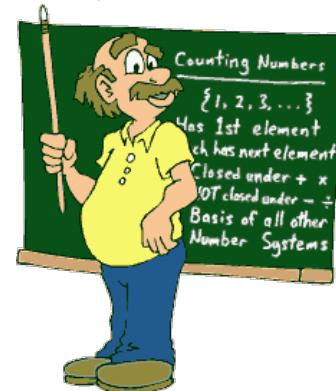


MSYS

Microcontroller Systems

Lektion 14: Timers - CTC mode

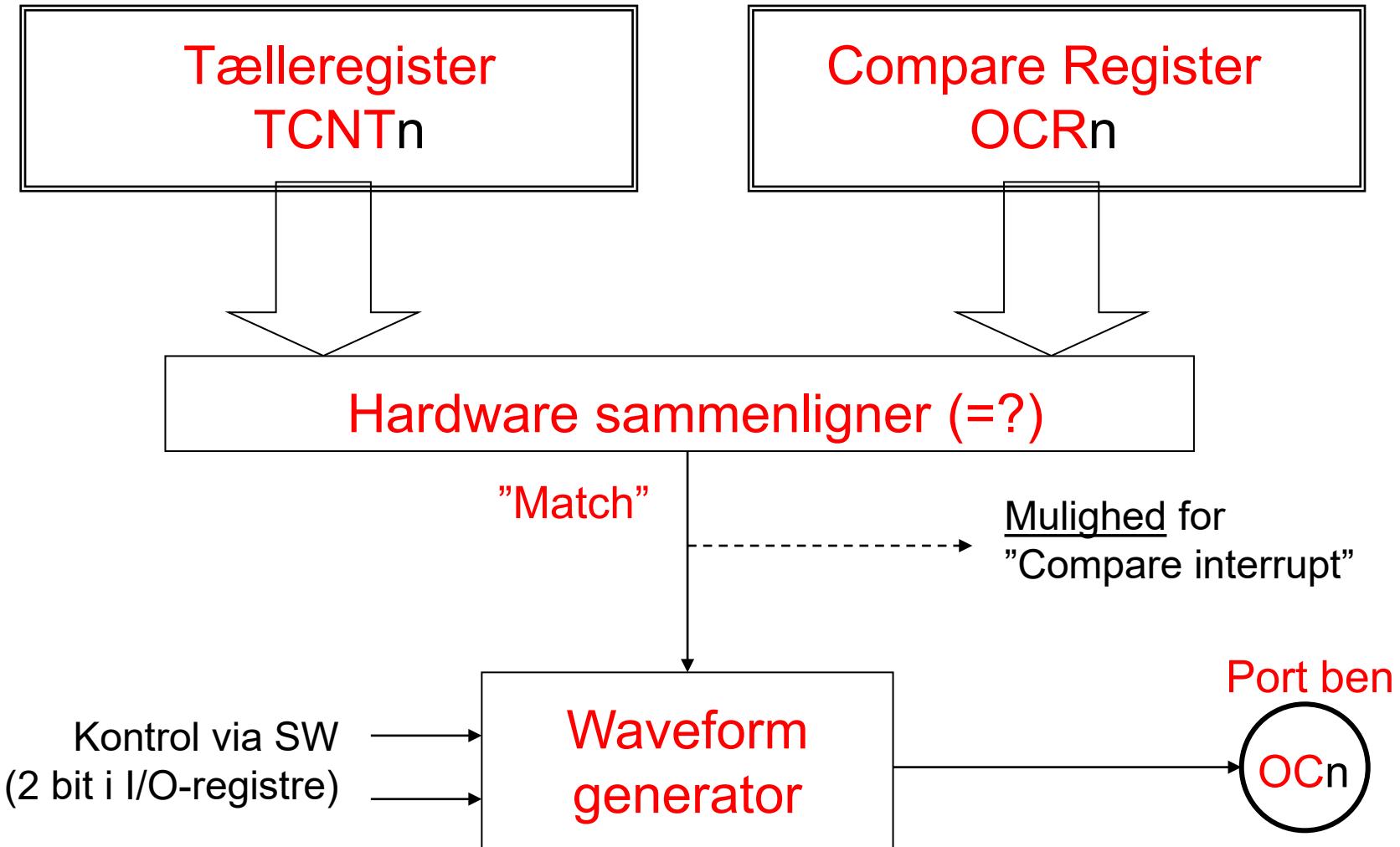
(Clear Timer on Compare match Mode)



Slides from Henning Hargaard

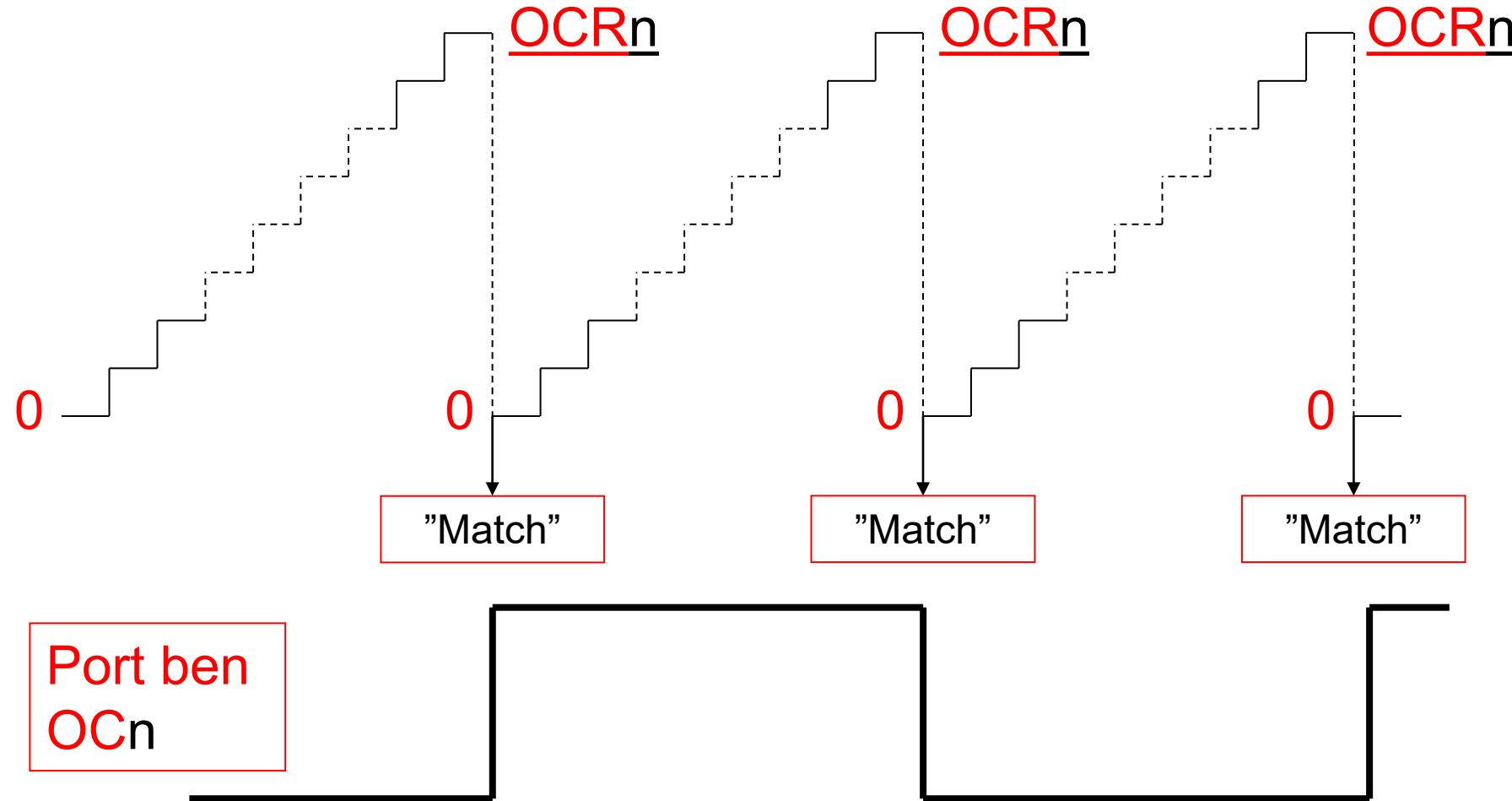
Michael Alrøe

Output Compare Unit



Kan sættes til automatisk at
toggle port benet ved match!

Timer i "CTC mode"



$$\text{Ben frekvens} = f_{\text{cpu}} / (2 * N * (1 + \text{OCRn}))$$

N er timerens prescaler-værdi

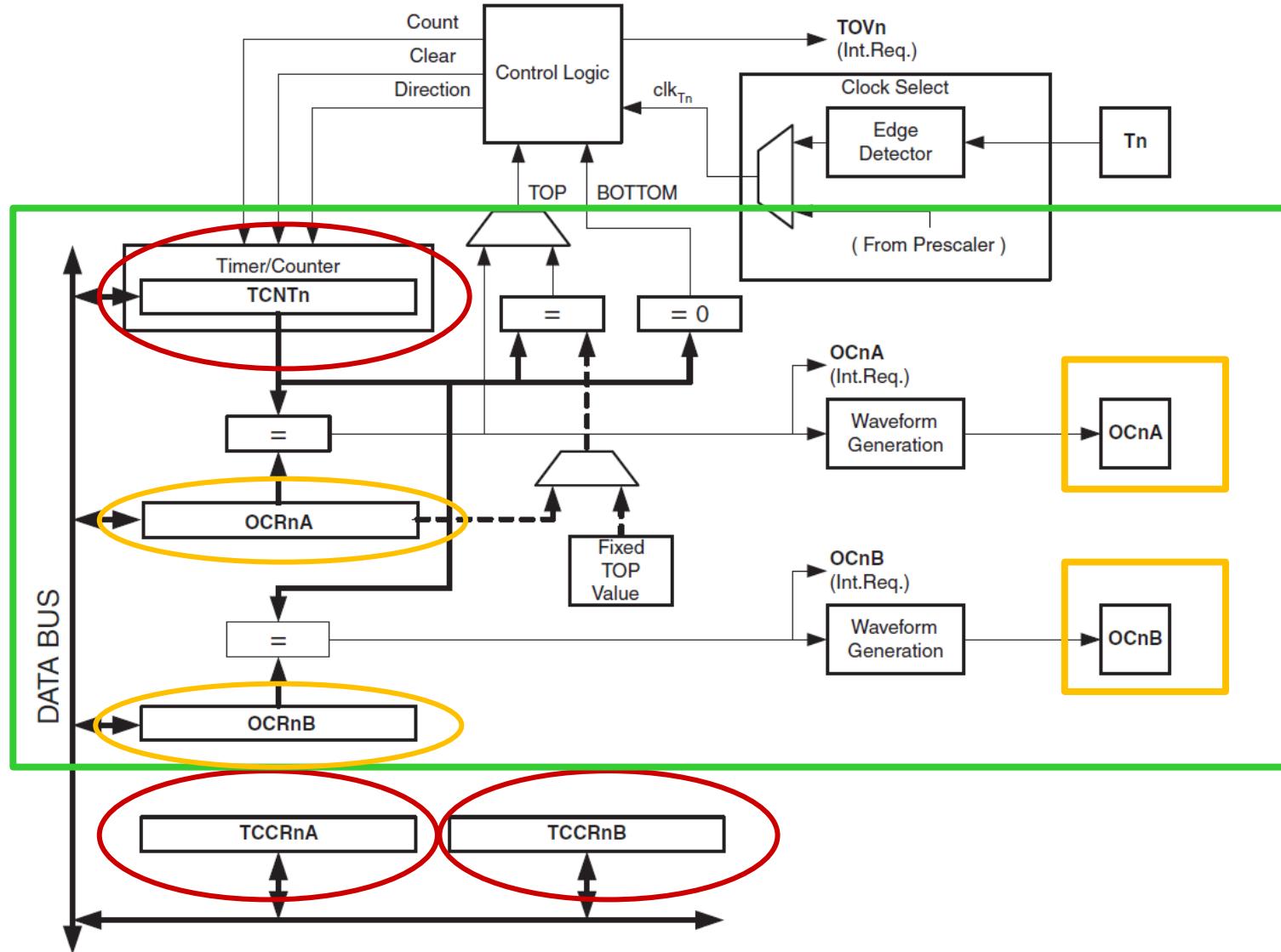
Mega32: 3 timere

- **Timer 0 :**
8 bit (MAX = 255).
Normal, CTC og PWM modes.
- **Timer 1 :**
16 bit (MAX = 65535).
Normal, CTC, mange PWM modes.
(Mulighed for "Input Capture")
- **Timer 2 :**
8 bit (MAX = 255).
Normal, CTC og PWM modes.
Asynkron mode (Real Time Clock).

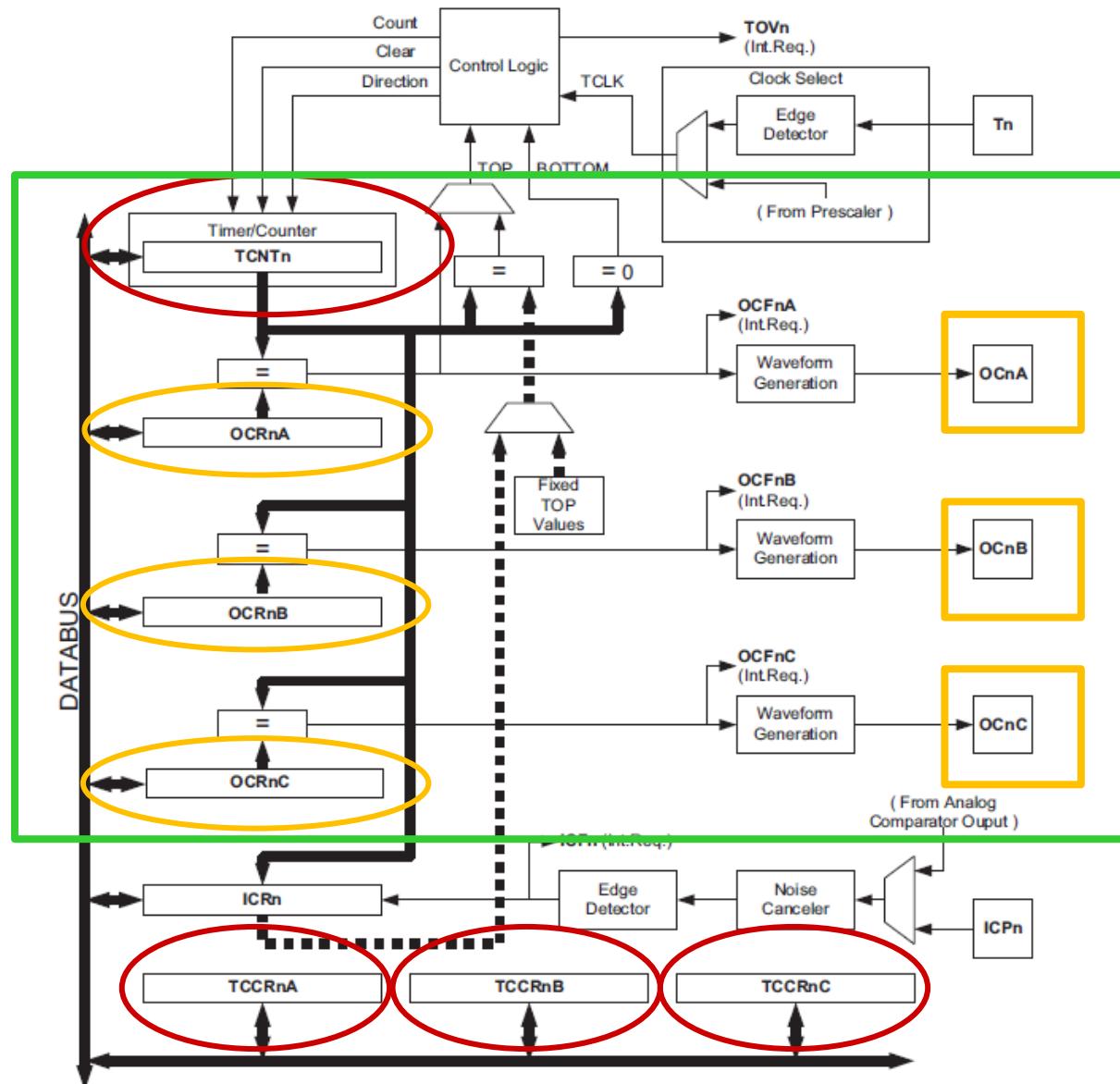
Mega2560: 6 timere

- Timer 0 :
8 bit (MAX = 255).
Normal, CTC og PWM modes.
- Timer 1, Timer 3, Timer 4 og Timer 5 :
16 bit (MAX = 65535).
Normal, CTC, mange PWM modes.
(Mulighed for "Input Capture")
- Timer 2 :
8 bit (MAX = 255).
Normal, CTC og PWM modes.
Asynkron mode (Real Time Clock).

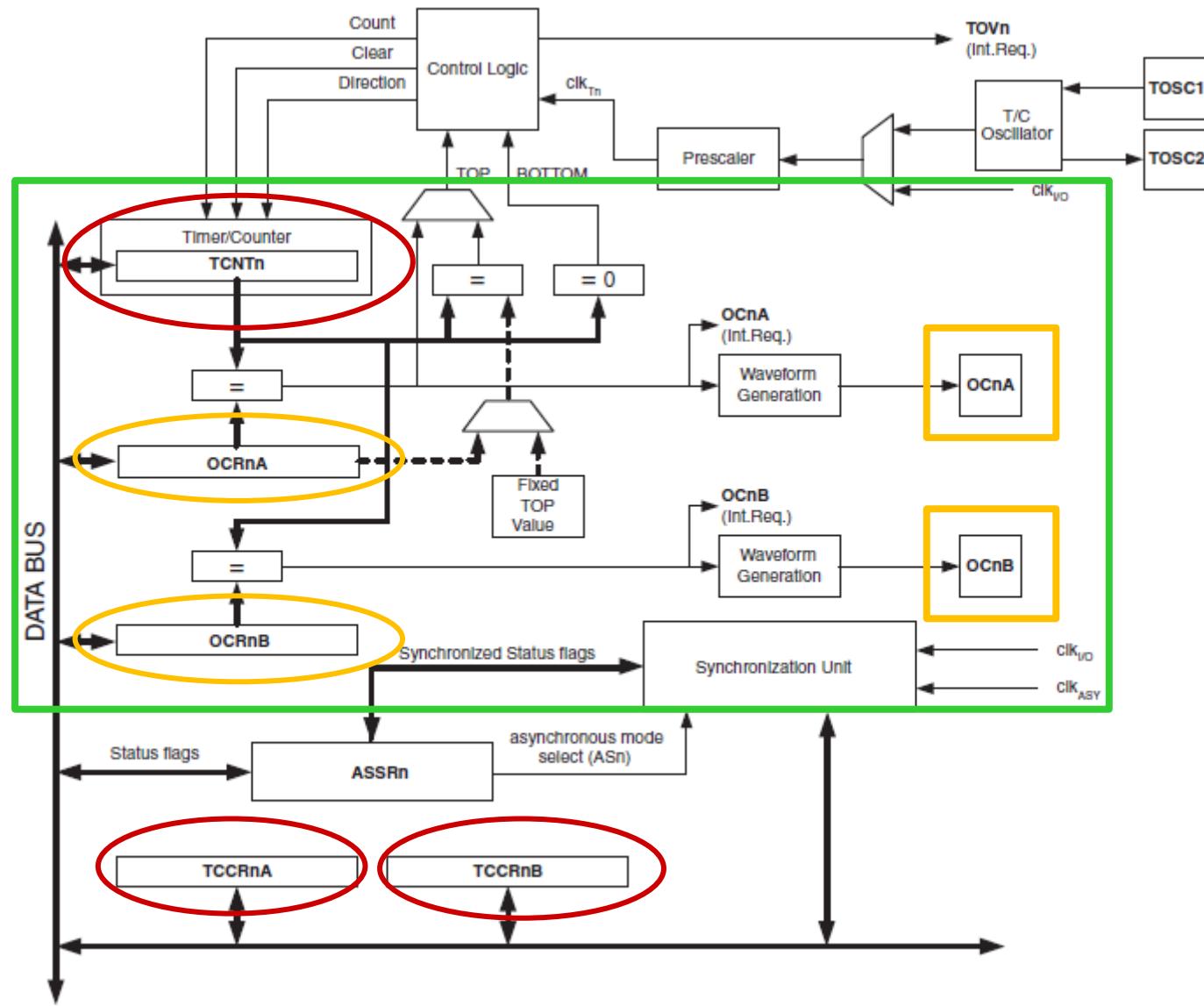
Mega2560: Timer 0 (8 bit)



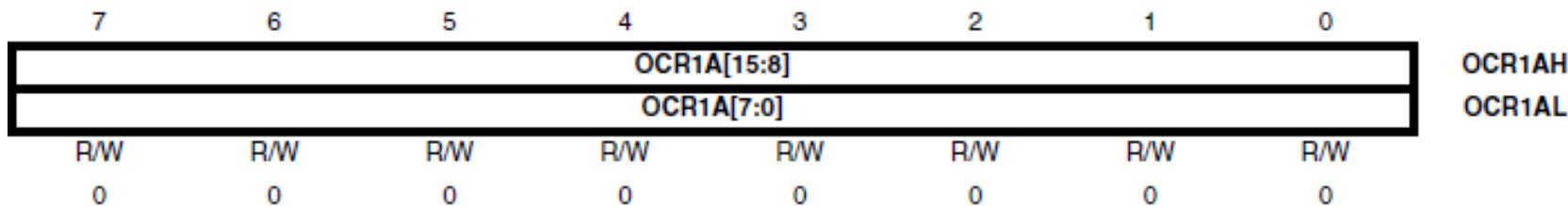
Mega2560: Timer 1,3,4,5 (16 bit)



Mega2560: Timer 2 (8 bit)



Output Compare Registre (16 bit)



```
AVR GCC C:  
#include <avr/io.h>  
  
// Herefter er 16-bit adgang muligt:  
OCR1A = 12345;
```

Valg af CTC Mode

- CTC mode vælges normalt under opstart (initiering).
- Hvilke registre, der skal skrives til, afhænger af, om vi bruger Mega32 eller Mega2560.
Desuden afhænger det af, hvilken timer, der drejer sig om.

Mega2560: Timer 0. CTC mode.

7	6	5	4	3	2	1	0	
COM0A1	COM0Ao	COM0B1	COM0B0	-	-	WGM01	WGM00	TCCR0A
R/W	R/W	R/W	R/W	R	R	R/W	R/W	
0	0	0	0	0	0	0	0	

7	6	5	4	3	2	1	0	
FOCoA	FOCoB	-	-	WGM02	CS02	CS01	CS00	TCCR0B
W	W	R	R	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	

Mode	WGM2	WGM1	WGM0	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on ⁽¹⁾⁽²⁾
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	TOP	MAX

Mega2560: Timer 1,3,4,5. CTC mode.



- TCCRnA = TCCR1A, TCCR3A, TCCR4A eller TCCR5A.
- TCCRnB = TCCR1B, TCCR3B, TCCR4B eller TCCR5B.

Mega2560: Timer 1,3,4,5. CTC mode.

Mode	WGMn3	WGMn2 (CTCn)	WGMn1 (PWMin1)	WGMn0 (PWMin0)	Timer/Counter Mode of Operation	TOP	Update of OCRnx at	TOVn Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCRnA	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICRn	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCRnA	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICRn	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCRnA	TOP	BOTTOM
12	1	1	0	0	CTC	ICRn	Immediate	MAX
13	1	1	0	1	(Reserved)	-	-	-
14	1	1	1	0	Fast PWM	ICRn	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCRnA	BOTTOM	TOP

OBS: I denne mode styres TOP (og dermed frekvensen)
af 16 bit registeret ICRn !

Mega2560: Timer 2. CTC mode.

7	6	5	4	3	2	1	0	
COM2A1	COM2A0	COM2B1	COM2B0	-	-	WGM21	WGM20	TCCR2A
R/W	R/W	R/W	R/W	R	R	R/W	R/W	

0 0 0 0 0 0 0 0

7	6	5	4	3	2	1	0	
FOC2A	FOC2B	-	-	WGM22	CS22	CS21	CS20	TCCR2B
W	W	R	R	R/W	R/W	R/W	R/W	

0 0 0 0 0 0 0 0

Mode	WGM2	WGM1	WGM0	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on ⁽¹⁾⁽²⁾
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX

Valg af clock

- De næste slides er de samme som fra "Normal mode".
- Gentaget for overskuelighedens skyld.

Mega2560: Timer 0. Valg af clock

7	6	5	4	3	2	1	0
FOC0A	FOC0B	-	-	WGM02	CS02	CS01	CS00
W	W	R	R	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$\text{clk}_{\text{I/O}}$ /(No prescaling)
0	1	0	$\text{clk}_{\text{I/O}}/8$ (From prescaler)
0	1	1	$\text{clk}_{\text{I/O}}/64$ (From prescaler)
1	0	0	$\text{clk}_{\text{I/O}}/256$ (From prescaler)
1	0	1	$\text{clk}_{\text{I/O}}/1024$ (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

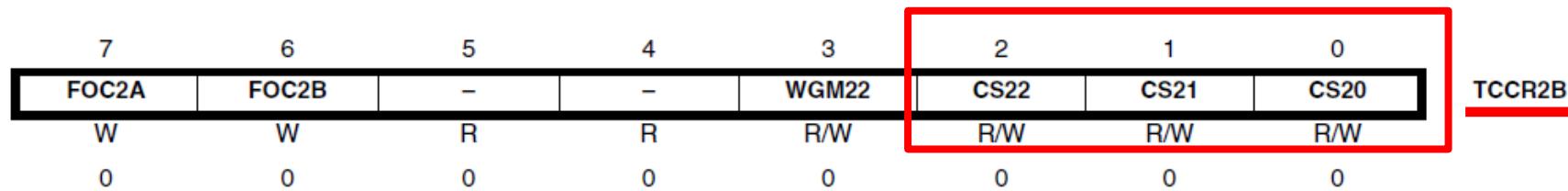
Mega2560: Timer 1,3,4,5. Valg af clock

7	6	5	4	3	2	1	0	
ICNC1	ICES1	-	WGM13	WGM12	CSn2	CSn1	CSn0	TCCRnB
R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	

- TCCRnB = TCCR1B, TCCR3B, TCCR4B eller TCCR5B.

CSn2	CSn1	CSn0	Description
0	0	0	No clock source. (Timer/Counter stopped)
0	0	1	$\text{clk}_{\text{I/O}}/1$ (No prescaling)
0	1	0	$\text{clk}_{\text{I/O}}/8$ (From prescaler)
0	1	1	$\text{clk}_{\text{I/O}}/64$ (From prescaler)
1	0	0	$\text{clk}_{\text{I/O}}/256$ (From prescaler)
1	0	1	$\text{clk}_{\text{I/O}}/1024$ (From prescaler)
1	1	0	External clock source on Tn pin. Clock on falling edge
1	1	1	External clock source on Tn pin. Clock on rising edge

Mega2560: Timer 2, Valg af clock



CS22	CS21	CS20	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$\text{clk}_{T2S}/(\text{No prescaling})$
0	1	0	$\text{clk}_{T2S}/8$ (From prescaler)
0	1	1	$\text{clk}_{T2S}/32$ (From prescaler)
1	0	0	$\text{clk}_{T2S}/64$ (From prescaler)
1	0	1	$\text{clk}_{T2S}/128$ (From prescaler)
1	1	0	$\text{clk}_{T2S}/256$ (From prescaler)
1	1	1	$\text{clk}_{T2S}/1024$ (From prescaler)

NYT: Pin styring i CTC Mode

- OC – benene styres af waveform-generatoren i CTC mode.
- Normalt vil vi vælge "Toggle pin on compare match".
- Se de næste slides for detaljer.

Mega2560: Timer 0. Pin styring A + B

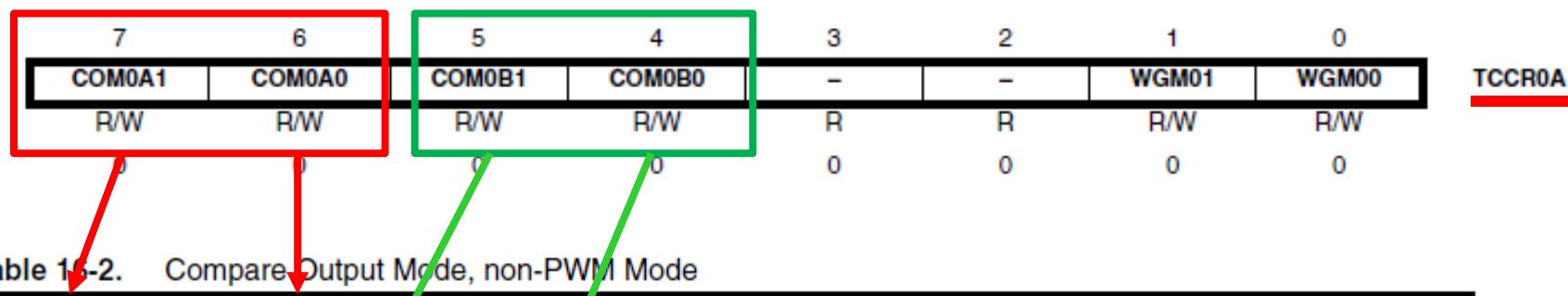


Table 16-2. Compare Output Mode, non-PWM Mode

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected
0	1	Toggle OC0A on Compare Match
1	0	Clear OC0A on Compare Match
1	1	Set OC0A on Compare Match

Table 16-5. Compare Output Mode, non-PWM Mode

COM0B1	COM0B0	Description
0	0	Normal port operation, OC0B disconnected
0	1	Toggle OC0B on Compare Match
1	0	Clear OC0B on Compare Match
1	1	Set OC0B on Compare Match

Mega2560: Timer 0. OC pins

OC0A = PB, ben 7

OC0B = PG, ben 5

Mega2560: Timer 1,3,4,5. Pin styring A+B+C

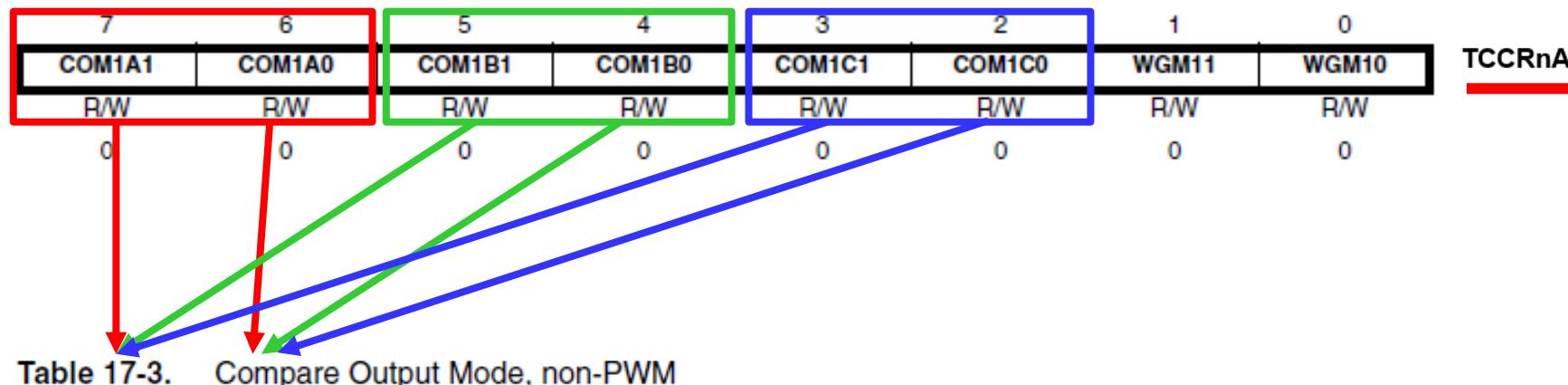


Table 17-3. Compare Output Mode, non-PWM

COMnA1 COMnB1 COMnC1	COMnA0 COMnB0 COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected
0	1	Toggle OCnA/OCnB/OCnC on compare match
1	0	Clear OCnA/OCnB/OCnC on compare match (set output to low level)
1	1	Set OCnA/OCnB/OCnC on compare match (set output to high level)

Rød = A-systemet.

Grøn = B-systemet.

Blå = C-systemet.

Mega2560: Timer 1,3,4,5. OC pins

Timer 1

OC1A = PB, ben 5

OC1B = PB, ben 6

OC1C = PB, ben 7

Timer 3

OC3A = PE, ben 3

OC3B = PE, ben 4

OC3C = PE, ben 5

Timer 4

OC4A = PH, ben 3

OC4B = PH, ben 4

OC4C = PH, ben 5

Timer 5

OC5A = PL, ben 3

OC5B = PL, ben 4

OC5C = PL, ben 5

Mega2560: Timer 2, Pin styring A + B

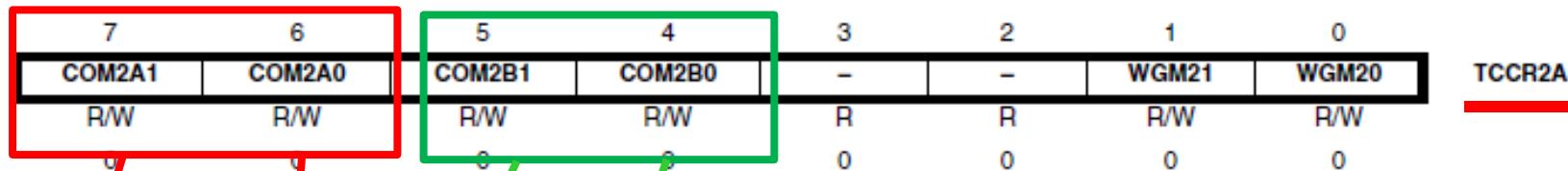


Table 20-2. Compare Output Mode, non-PWM Mode

COM2A1	COM2A0	Description
0	0	Normal port operation, OC2A disconnected
0	1	Toggle OC2A on Compare Match
1	0	Clear OC2A on Compare Match
1	1	Set OC2A on Compare Match

Table 20-5. Compare Output Mode, non-PWM Mode

COM2B1	COM2B0	Description
0	0	Normal port operation, OC2B disconnected
0	1	Toggle OC2B on Compare Match
1	0	Clear OC2B on Compare Match
1	1	Set OC2B on Compare Match

Mega2560: Timer 2. OC pins

OC2A = PB, ben 4

OC2B = PH, ben 6

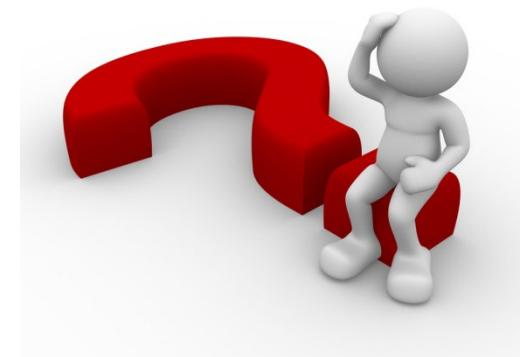
Test ("socrative.com"): Room = AUQUIZ

- En Mega2560's clockfrekvens er **3,6864 MHz**.
Timer 0 er initieret til CTC mode, og der er valgt "Toggle OC0A on compare match".
Timer 0's clock prescaler er sat til **1024**.
Hvilken frekvens vil kunne måles på benet OC0A, når register **OCR0A = 199** ?
- A: 18 Hz.
- B: 199 Hz.
- C: 1024 Hz.
- D: 9 Hz.



Test ("socrative.com"): Room = AUQUIZ

- Mega2560's Timer 1 er i CTC mode (mode 4), og CPU clock-frekvensen er **4 MHz**.
Timer 1's clock prescaler = **8**.
Hvilken værdi skal skrives til register OCR1A, for at man kan generere et firkantsignal på benet OC1A på **1000 Hz** ?
- A: $\text{OCR1A} = 1000$;
- B: $\text{OCR1A} = 4000000$;
- C: $\text{OCR1A} = 249$;
- D: $\text{OCR1A} = 999$;



Slut på lektion 14

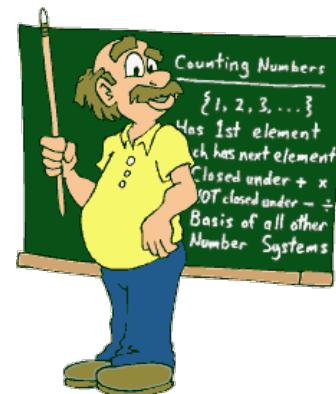


MSYS

Microcontroller Systems

Lektion 15: Timers - PWM mode

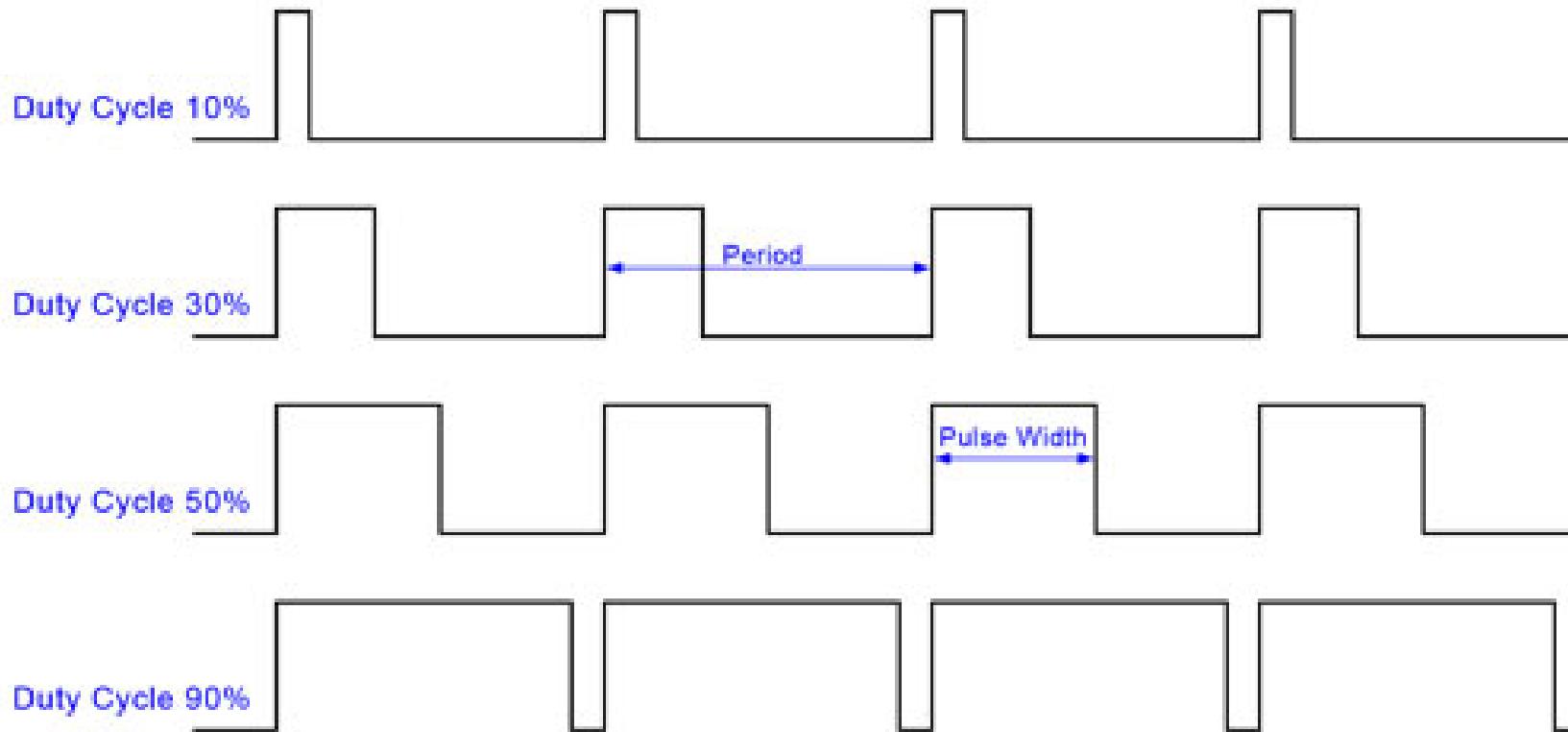
(Pulse Width Modulation Mode)



Slides from Henning Hargaard

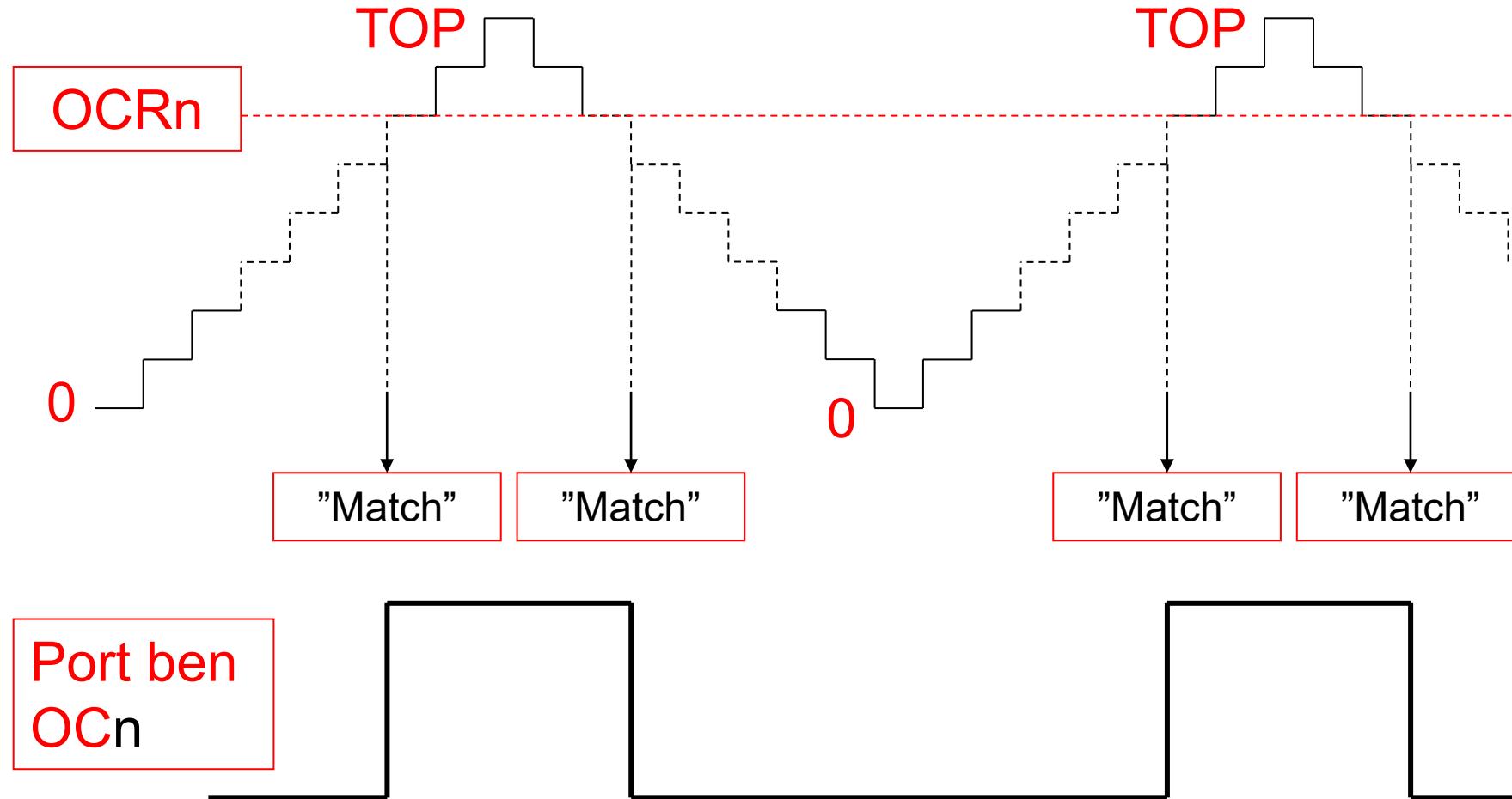
Michael Alrøe

Pulse Width Modulation (PWM)



$$\text{Duty Cycle} = (\text{Pulse Width} / \text{Period}) \times 100 [\%]$$

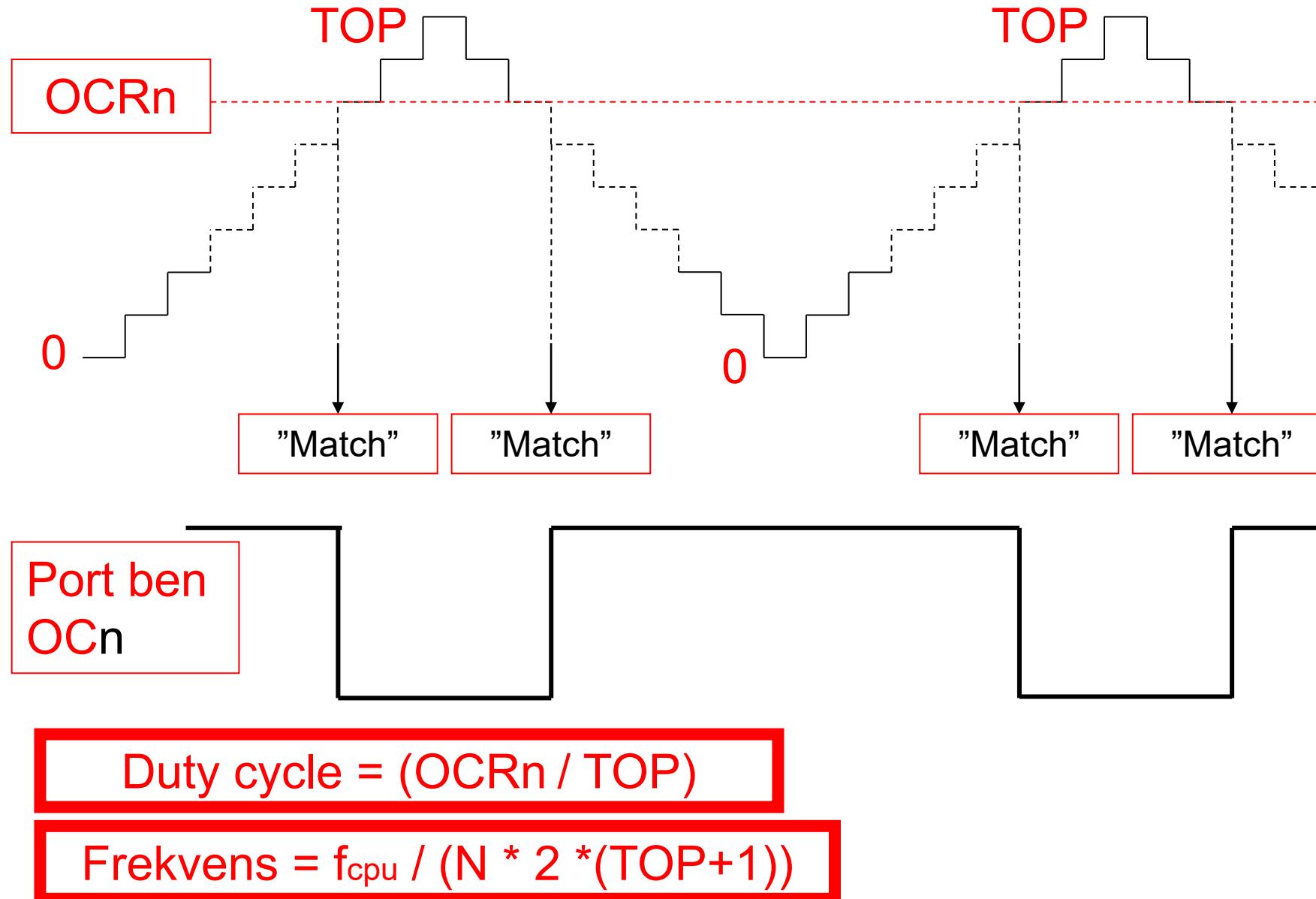
Timer i PWM mode (ikke "fast mode")



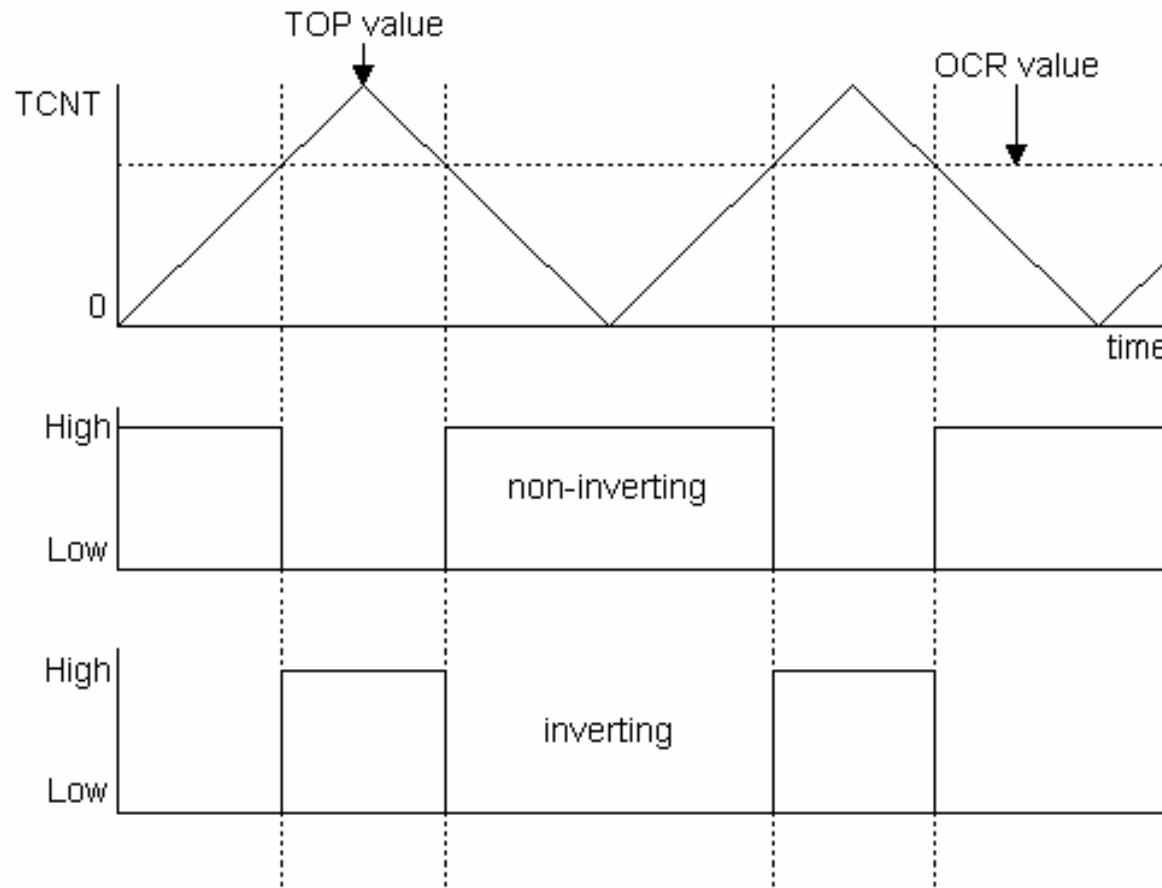
$$\text{Duty cycle} = 1 - (\text{OCRn} / \text{TOP})$$

$$\text{Frekvens} = f_{\text{cpu}} / (N * 2 * (\text{TOP}+1))$$

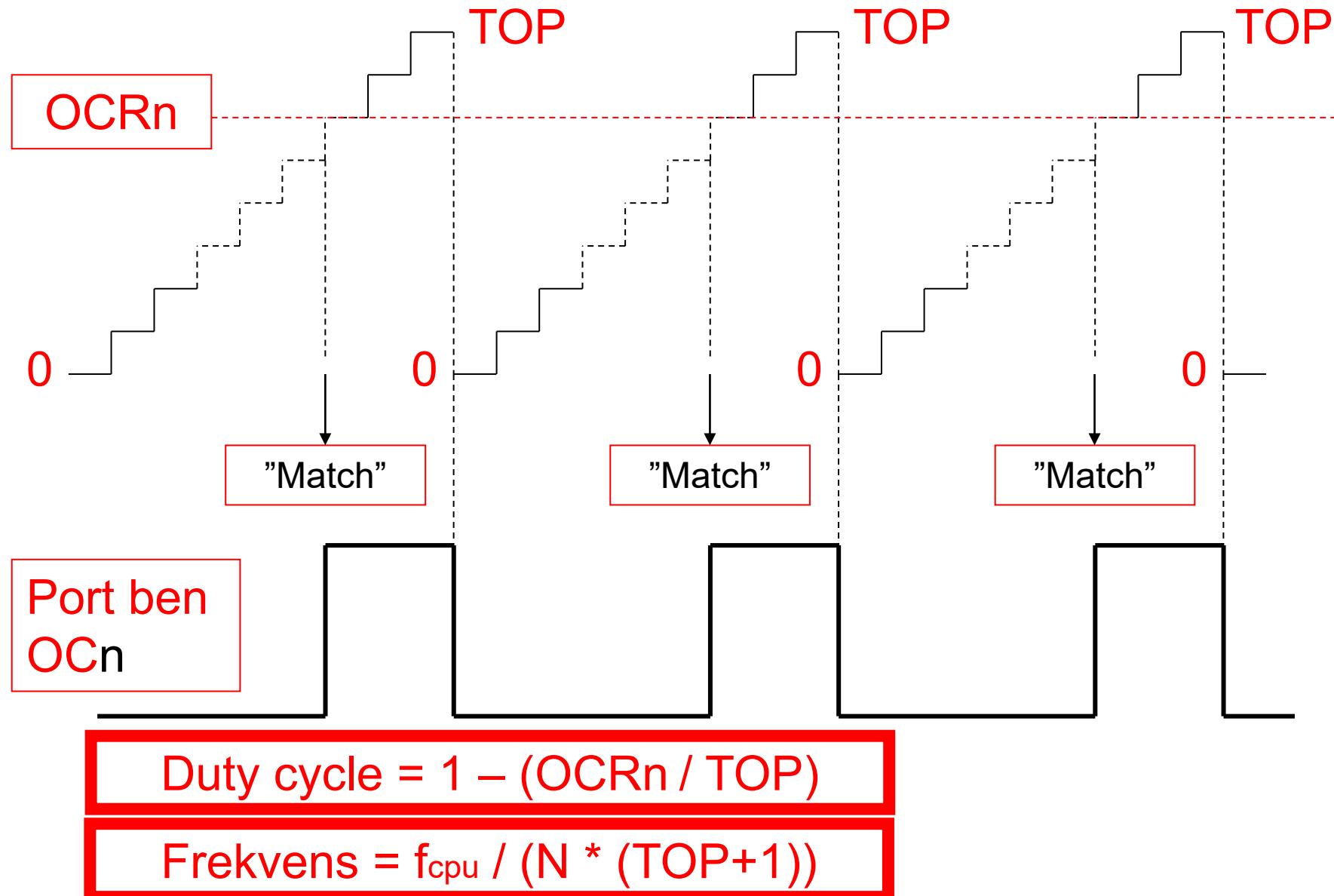
Timer i PWM mode (ikke "fast mode")



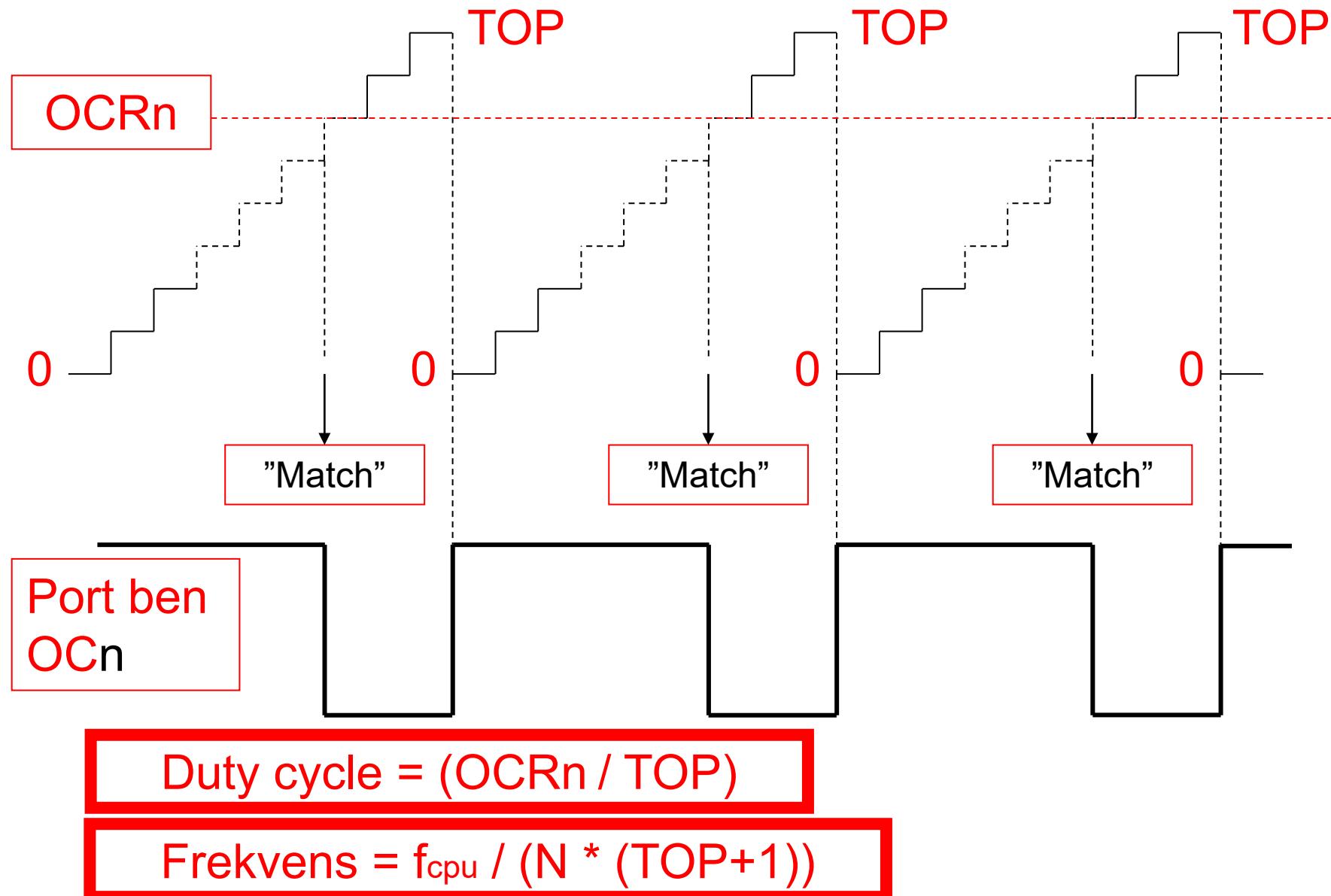
Output Compare og PWM (ikke "fast")



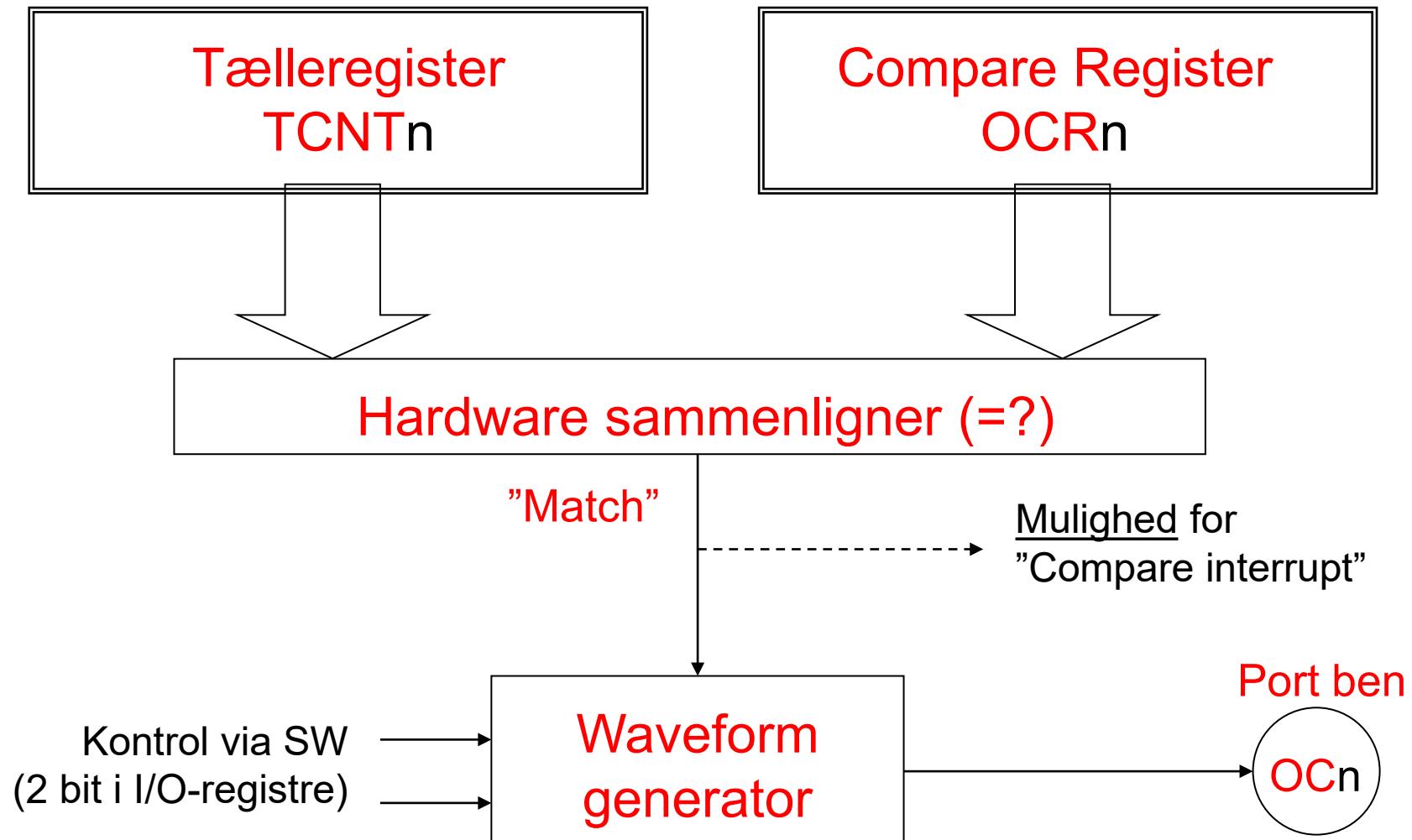
Timer i fast PWM mode



Timer i fast PWM mode



Output Compare Unit



Speciel funktion i PWM modes !

Blokdiagrammer

- De næste slides er de samme som fra ”CTC mode”.
- Gentaget for overskuelighedens skyld.

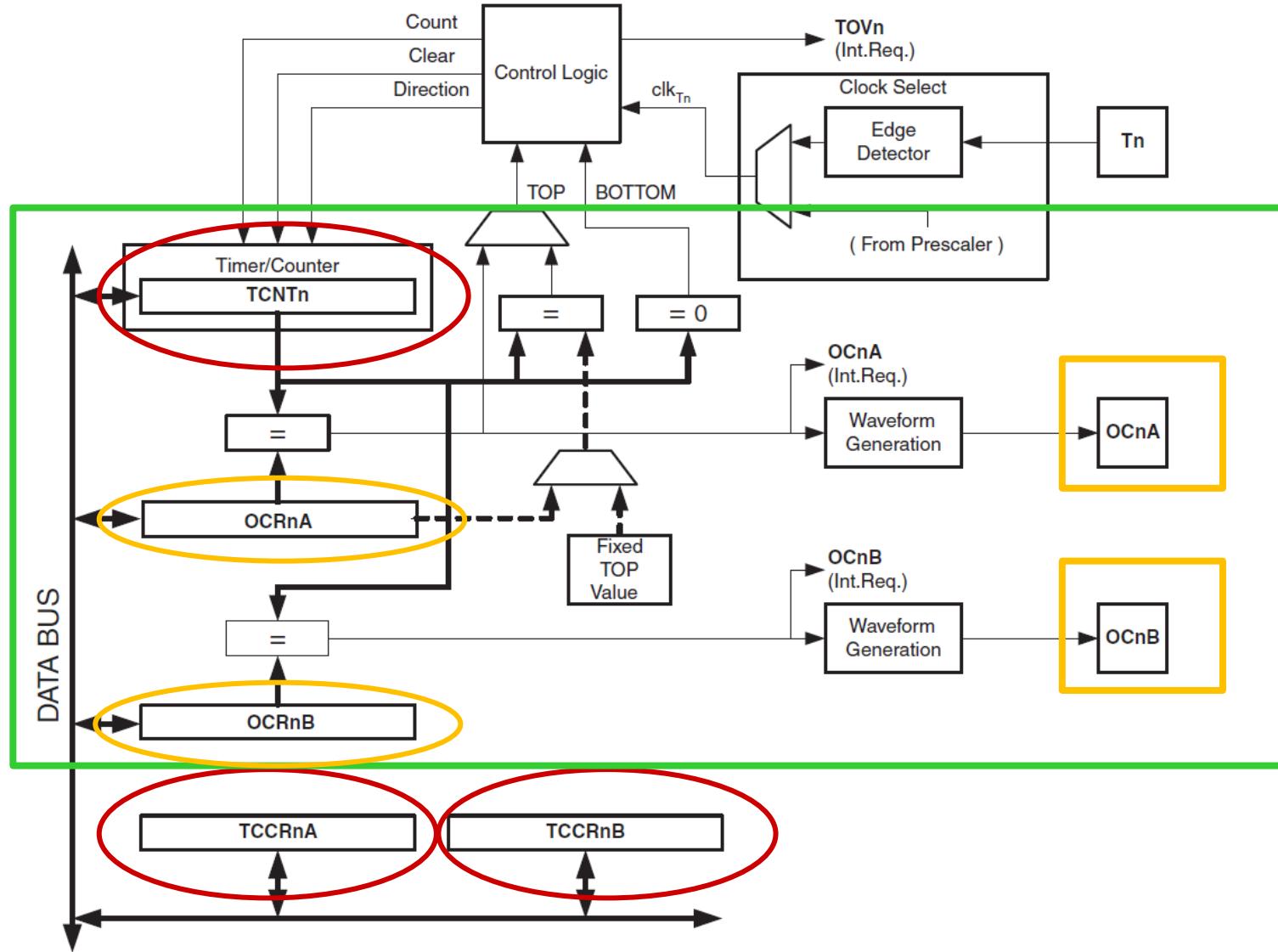
Mega32: 3 timere

- **Timer 0 :**
8 bit (MAX = 255).
Normal, CTC og PWM modes.
- **Timer 1 :**
16 bit (MAX = 65535).
Normal, CTC, mange PWM modes.
(Mulighed for "Input Capture")
- **Timer 2 :**
8 bit (MAX = 255).
Normal, CTC og PWM modes.
Asynkron mode (Real Time Clock).

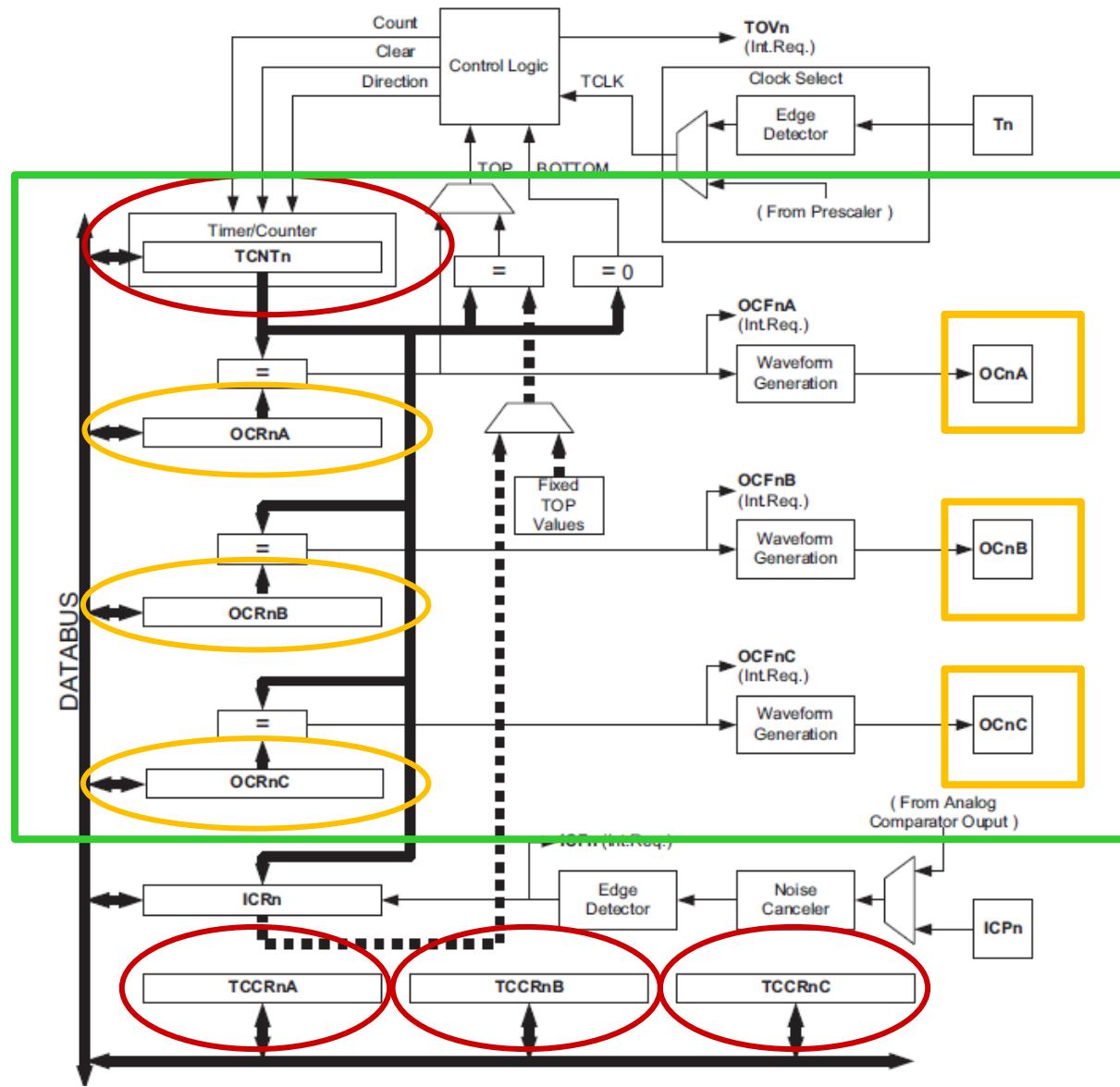
Mega2560: 6 timere

- **Timer 0 :**
8 bit (MAX = 255).
Normal, CTC og PWM modes.
- **Timer 1, Timer 3, Timer 4 og Timer 5 :**
16 bit (MAX = 65535).
Normal, CTC, mange PWM modes.
(Mulighed for "Input Capture")
- **Timer 2 :**
8 bit (MAX = 255).
Normal, CTC og PWM modes.
Asynkron mode (Real Time Clock).

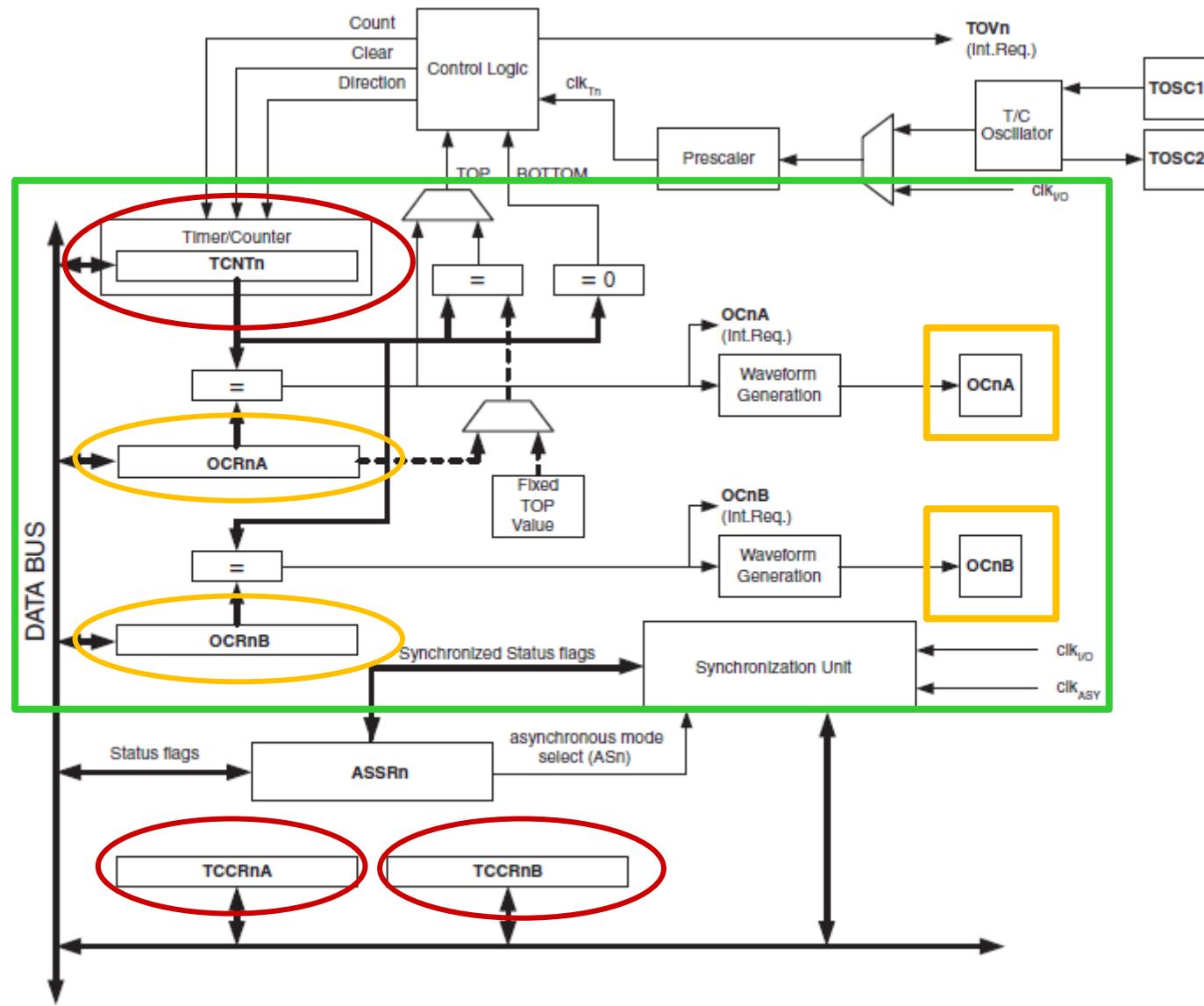
Mega2560: Timer 0 (8 bit)



Mega2560: Timer 1,3,4,5 (16 bit)



Mega2560: Timer 2 (8 bit)



NYT: Valg af PWM Mode

- PWM mode vælges normalt under opstart (initiering).
- Hvilke registre, der skal skrives til, afhænger af, om vi bruger Mega32 eller Mega2560.

Desuden afhænger det af, hvilken timer, der drejer sig om.

Mega2560: Timer 0. PWM modes

7	6	5	4	3	2	1	0	
COMoA1	COMoAo	COMoB1	COMoBo	-	-	WGM01	WGM00	TCCR0A
R/W	R/W	R/W	R/W	R	R	R/W	R/W	
0	0	0	0	0	0	0	0	
7	6	5	4	3	2	1	0	
FOCoA	FOCoB	-	-	WGM02	CS02	CS01	CS00	TCCR0B
W	W	R	R	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	

Mode	WGM2	WGM1	WGM0	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on ⁽¹⁾⁽²⁾
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	TOP	MAX
4	1	0	0	Reserved	-	-	-
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	-	-	-
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

OBS : TOP er afhængig af mode !

Mega2560: Timer 1,3,4,5. PWM modes



- TCCR_nA = TCCR1A, TCCR3A, TCCR4A eller TCCR5A.
- TCCR_nB = TCCR1B, TCCR3B, TCCR4B eller TCCR5B.

Mega2560: Timer 1,3,4,5. PWM modes

Mode	WG _{Mn3}	WG _{Mn2} (CTC _n)	WG _{Mn1} (PWM _{n1})	WG _{Mn0} (PWM _{n0})	Timer/Counter Mode of Operation	TOP	Update of OCR _{RnX} at	TOV _n Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR _{RnA}	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR _n	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR _{RnA}	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR _n	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR _{RnA}	TOP	BOTTOM
12	1	1	0	0	CTC	ICR _n	Immediate	MAX
13	1	1	0	1	(Reserved)	-	-	-
14	1	1	1	0	Fast PWM	ICR _n	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR _{RnA}	BOTTOM	TOP

OBS : TOP er afhængig af mode !

Mega2560: Timer 2. PWM modes

7	6	5	4	3	2	1	0	
COM2A1	COM2A0	COM2B1	COM2B0	-	-	WGM21	WGM20	TCCR2A
R/W	R/W	R/W	R/W	R	R	R/W	R/W	
0	0	0	0	0	0	0	0	
7	6	5	4	3	2	1	0	
FOC2A	FOC2B	-	-	WGM22	CS22	CS21	CS20	TCCR2B
W	W	R	R	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	

Mode	WGM2	WGM1	WGM0	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on ⁽¹⁾⁽²⁾
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	-	-	-
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	-	-	-
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

OBS : TOP er afhængig af mode !



Valg af clock

- De næste slides er de samme som fra "Normal mode" og "CTC mode".
- Gentaget for overskuelighedens skyld.

Mega2560: Timer 0. Valg af clock

7	6	5	4	3	2	1	0
FOC0A	FOC0B	-	-	WGM02	CS02	CS01	CS00
W	W	R	R	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

TCCR0B

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$\text{clk}_{\text{I/O}}$ /(No prescaling)
0	1	0	$\text{clk}_{\text{I/O}}/8$ (From prescaler)
0	1	1	$\text{clk}_{\text{I/O}}/64$ (From prescaler)
1	0	0	$\text{clk}_{\text{I/O}}/256$ (From prescaler)
1	0	1	$\text{clk}_{\text{I/O}}/1024$ (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

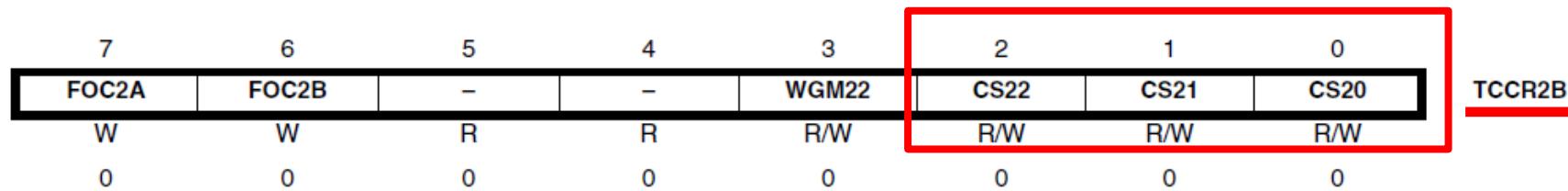
Mega2560: Timer 1,3,4,5. Valg af clock

7	6	5	4	3	2	1	0	
ICNC1	ICES1	-	WGM13	WGM12	CSn2	CSn1	CSn0	TCCRnB
R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	

- TCCRnB = TCCR1B, TCCR3B, TCCR4B eller TCCR5B.

CSn2	CSn1	CSn0	Description
0	0	0	No clock source. (Timer/Counter stopped)
0	0	1	$\text{clk}_{\text{I/O}}/1$ (No prescaling)
0	1	0	$\text{clk}_{\text{I/O}}/8$ (From prescaler)
0	1	1	$\text{clk}_{\text{I/O}}/64$ (From prescaler)
1	0	0	$\text{clk}_{\text{I/O}}/256$ (From prescaler)
1	0	1	$\text{clk}_{\text{I/O}}/1024$ (From prescaler)
1	1	0	External clock source on Tn pin. Clock on falling edge
1	1	1	External clock source on Tn pin. Clock on rising edge

Mega2560: Timer 2. Valg af clock

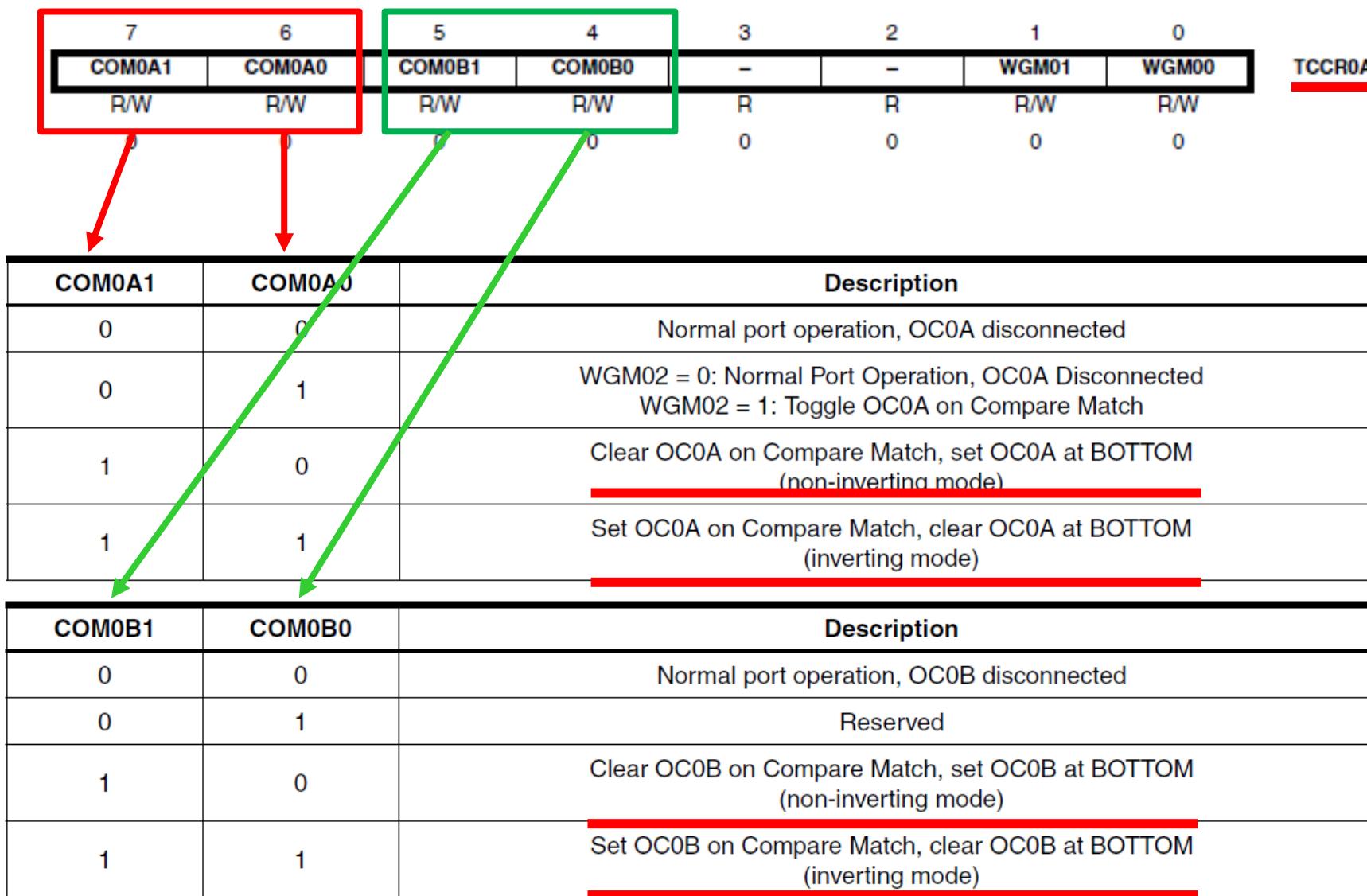


CS22	CS21	CS20	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$\text{clk}_{T2S}/(\text{No prescaling})$
0	1	0	$\text{clk}_{T2S}/8$ (From prescaler)
0	1	1	$\text{clk}_{T2S}/32$ (From prescaler)
1	0	0	$\text{clk}_{T2S}/64$ (From prescaler)
1	0	1	$\text{clk}_{T2S}/128$ (From prescaler)
1	1	0	$\text{clk}_{T2S}/256$ (From prescaler)
1	1	1	$\text{clk}_{T2S}/1024$ (From prescaler)

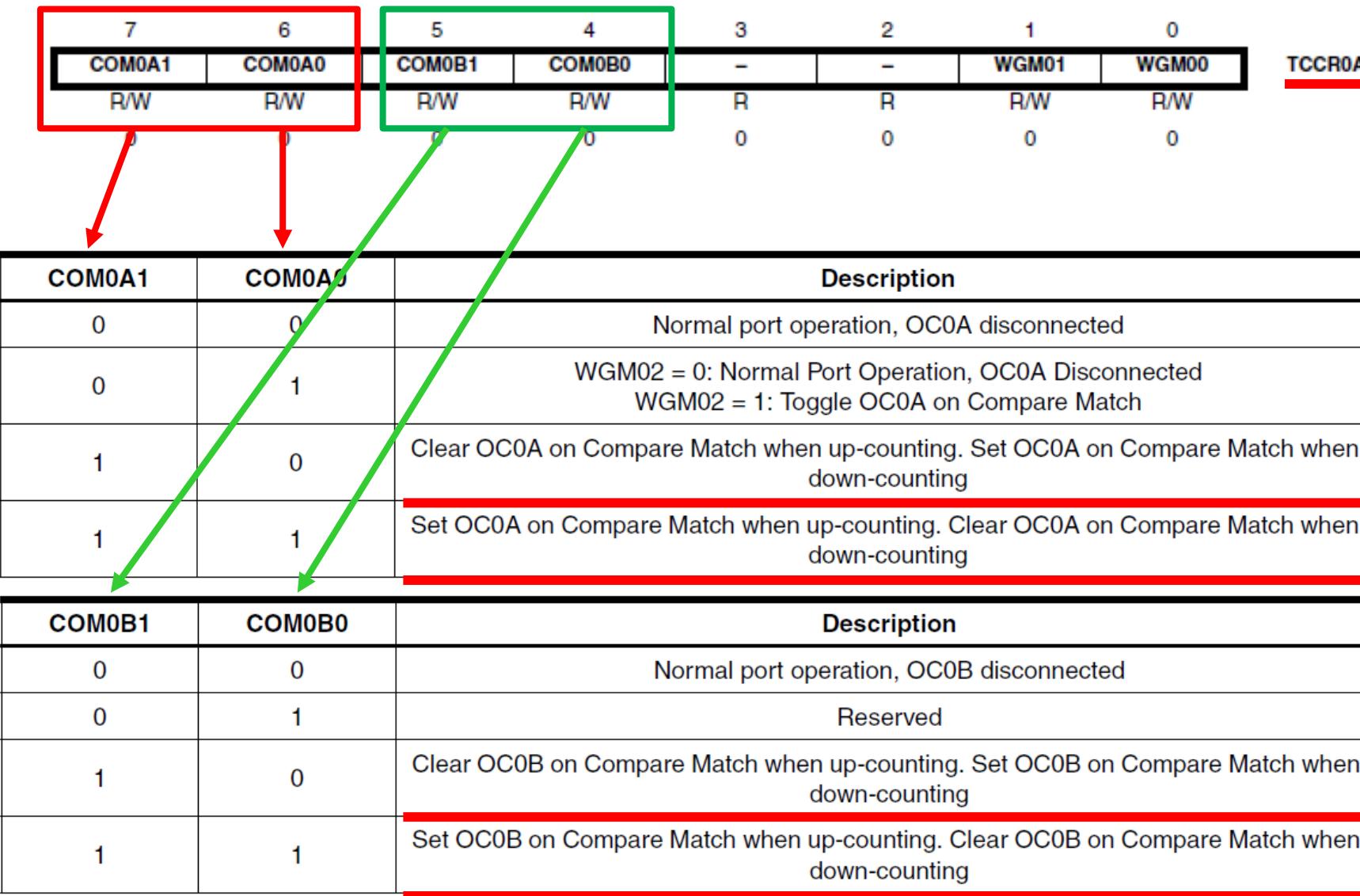
NYT: Pin styring i PWM Mode

- OC – benene styres af waveform-generatoren i PWM mode.
- Der er forskel på, hvilken ”tabel” der skal bruges for ”fast PWM mode” og ”ikke-fast PWM mode”.
- Pas derfor på at anvende den rigtige ”tabel” (se de næste slides).

Mega2560: Timer 0. Pin styring A + B. Fast PWM.



Mega2560: Timer 0. Pin styring A + B. Ikke-fast PWM.

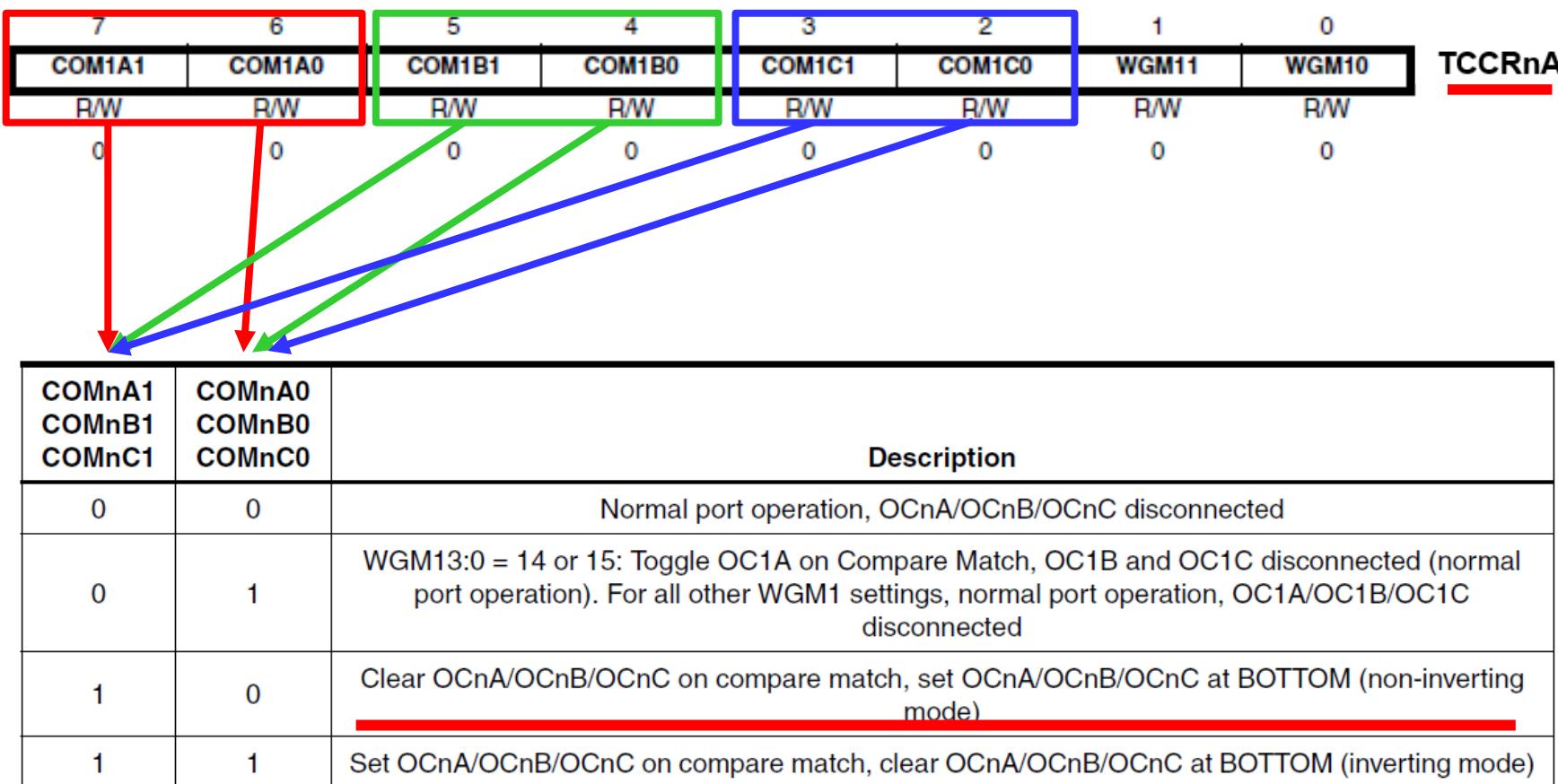


Mega2560: Timer 0. OC pins.

OC0A = PB, ben 7

OC0B = PG, ben 5

Mega2560: Timer 1,3,4,5. Pin styring A+B+C. Fast PWM.

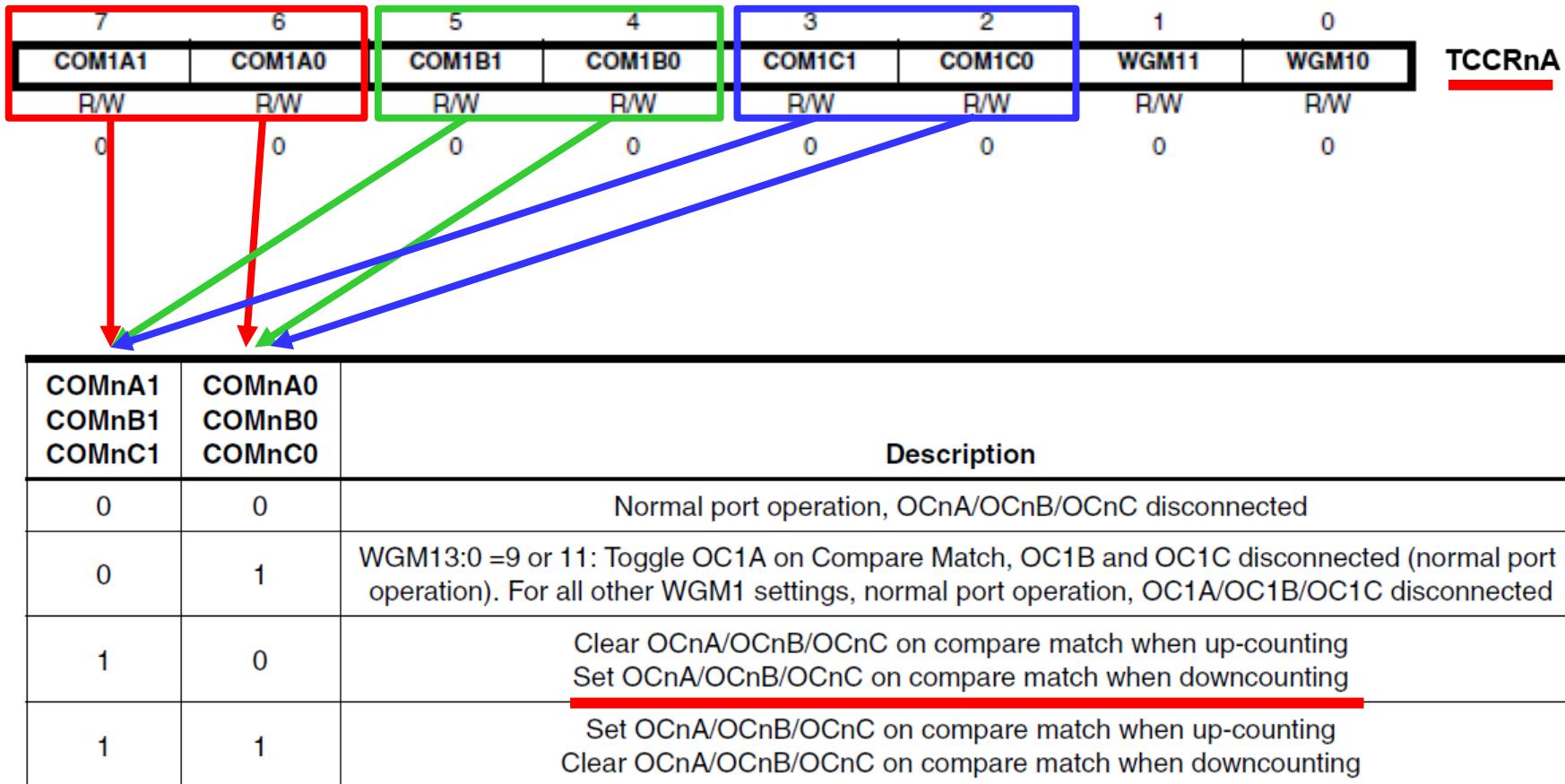


Rød = A-systemet.

Grøn = B-systemet.

Blå = C-systemet.

Mega2560: Timer 1,3,4,5. Pin styring A+B+C. Ikke-fast PWM.



Rød = A-systemet.

Grøn = B-systemet.

Blå = C-systemet.

Mega2560: Timer 1,3,4,5. OC pins

Timer 1

OC1A = PB, ben 5

OC1B = PB, ben 6

OC1C = PB, ben 7

Timer 3

OC3A = PE, ben 3

OC3B = PE, ben 4

OC3C = PE, ben 5

Timer 4

OC4A = PH, ben 3

OC4B = PH, ben 4

OC4C = PH, ben 5

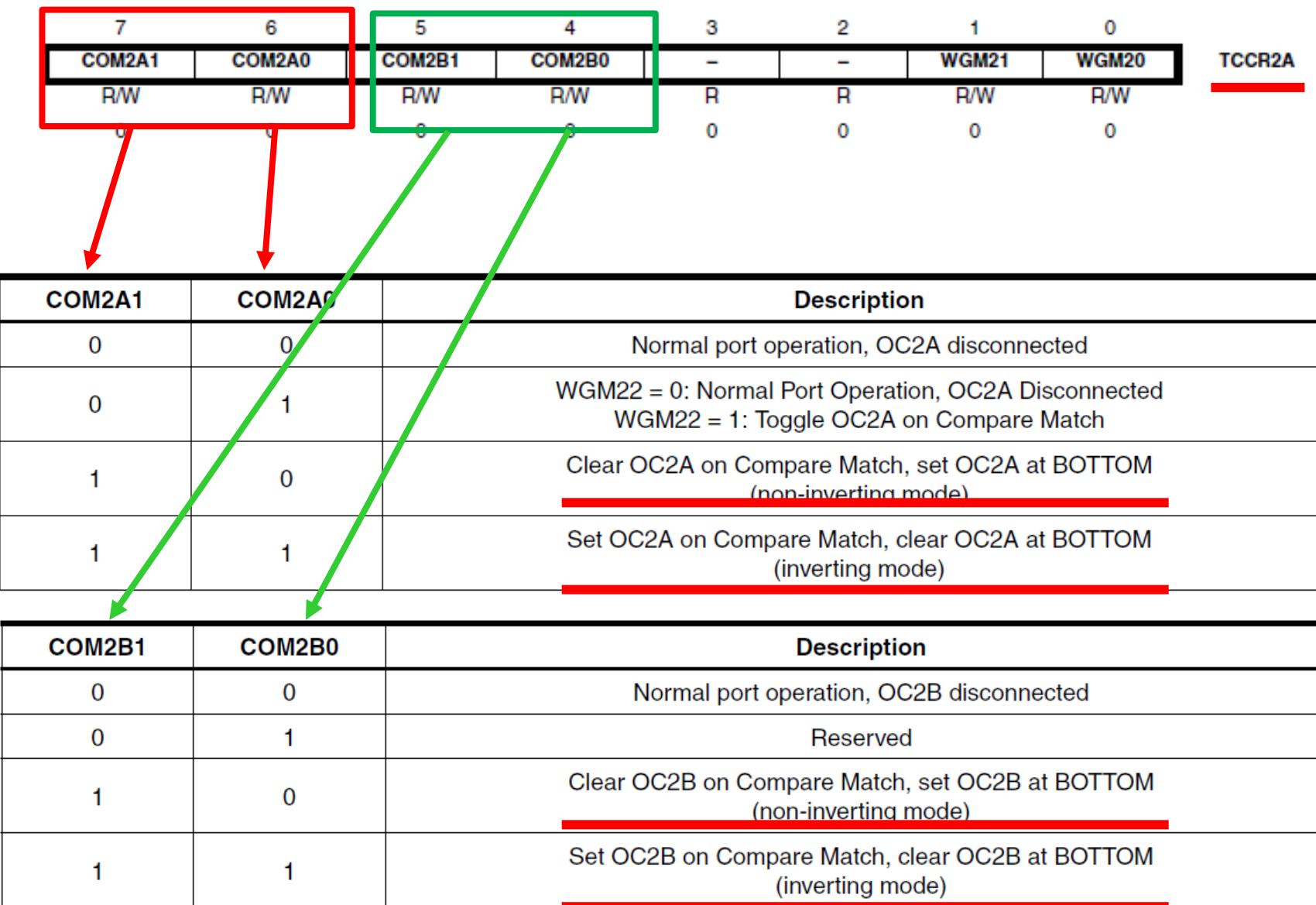
Timer 5

OC5A = PL, ben 3

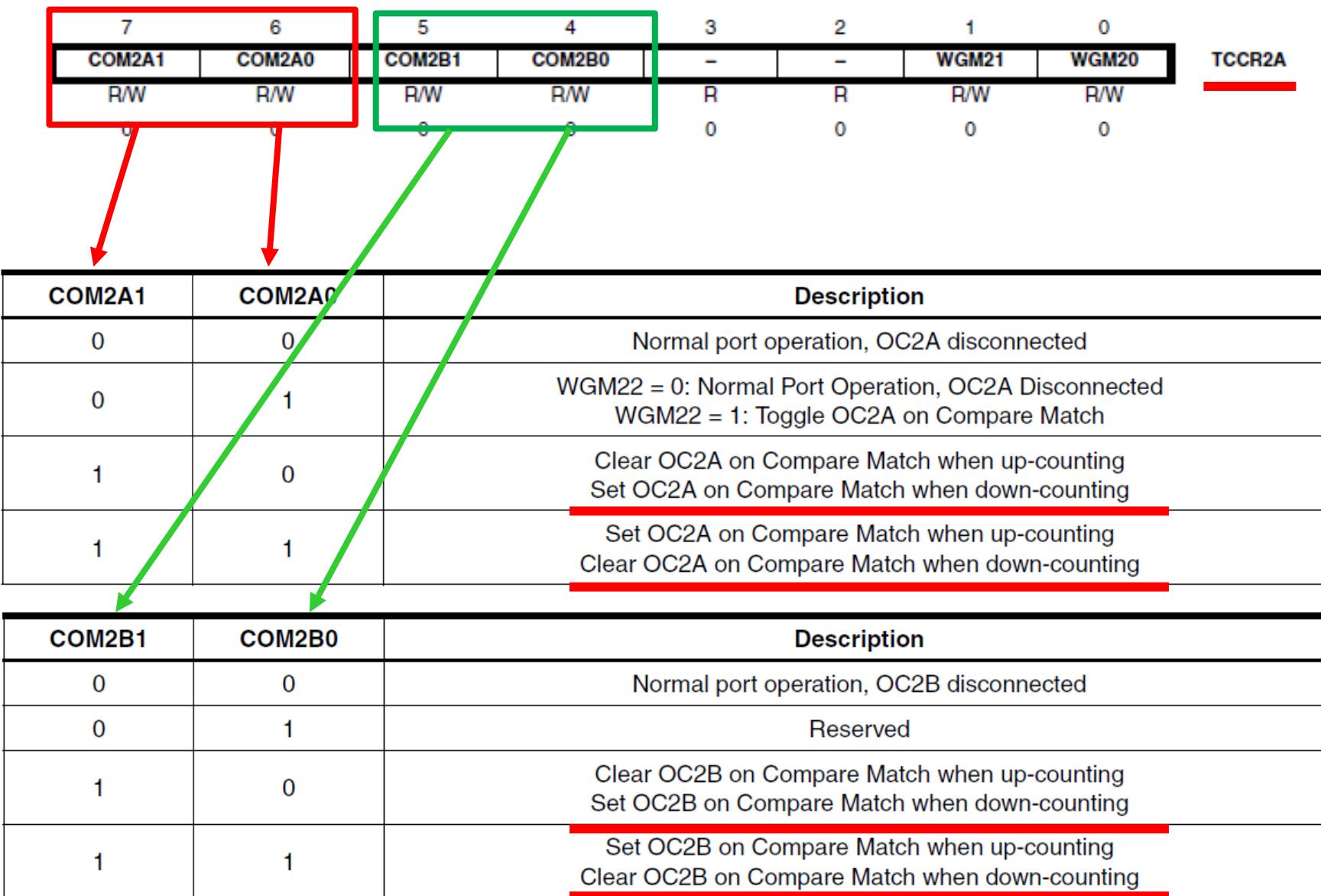
OC5B = PL, ben 4

OC5C = PL, ben 5

Mega2560: Timer 2. Pin styring A + B. Fast PWM.



Mega2560: Timer 2. Pin styring A+B. Ikke-fast PWM.



Mega2560: Timer 2. OC pins.

OC2A = PB, ben 4

OC2B = PH, ben 6

Test ("socrative.com"): Room = AUQUIZ

Denne kode initierer Mega2560's Timer 1:

DDRB = 0xFF;

TCCR1A = 0b11000011;

TCCR1B = 0b00000001;

Hvilken værdi skal herefter skrives til registeret OCR1A for at få et PWM signal på OC1A benet med **50% duty cycle?**

- A: OCR1A = 128;
- B: OCR1A = 256;
- C: OCR1A = 512;
- D: OCR1A = 1024;



Test ("socrative.com"): Room = AUQUIZ

Mega2560's Timer 1 er initieret til "**10 bit Phase Correct PWM**".

CPU clock-frekvens er **3,6864 MHz**.

Timer 1's clock prescaler er sat til **256**.

Hvilken frekvens har PWM-signalen ?

- A: Cirka 14 Hz
- B: Cirka 3600 Hz
- C: Cirka 7 Hz
- D: Cirka 1 Hz

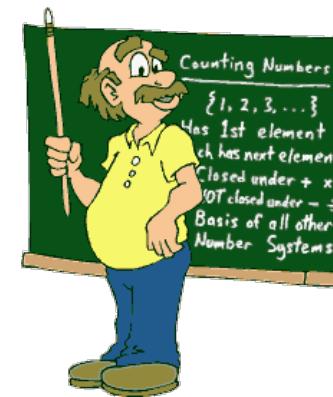
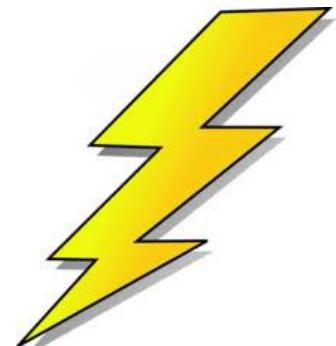




MSYS

Microcontroller Systems

Lektion 16: Interrupts

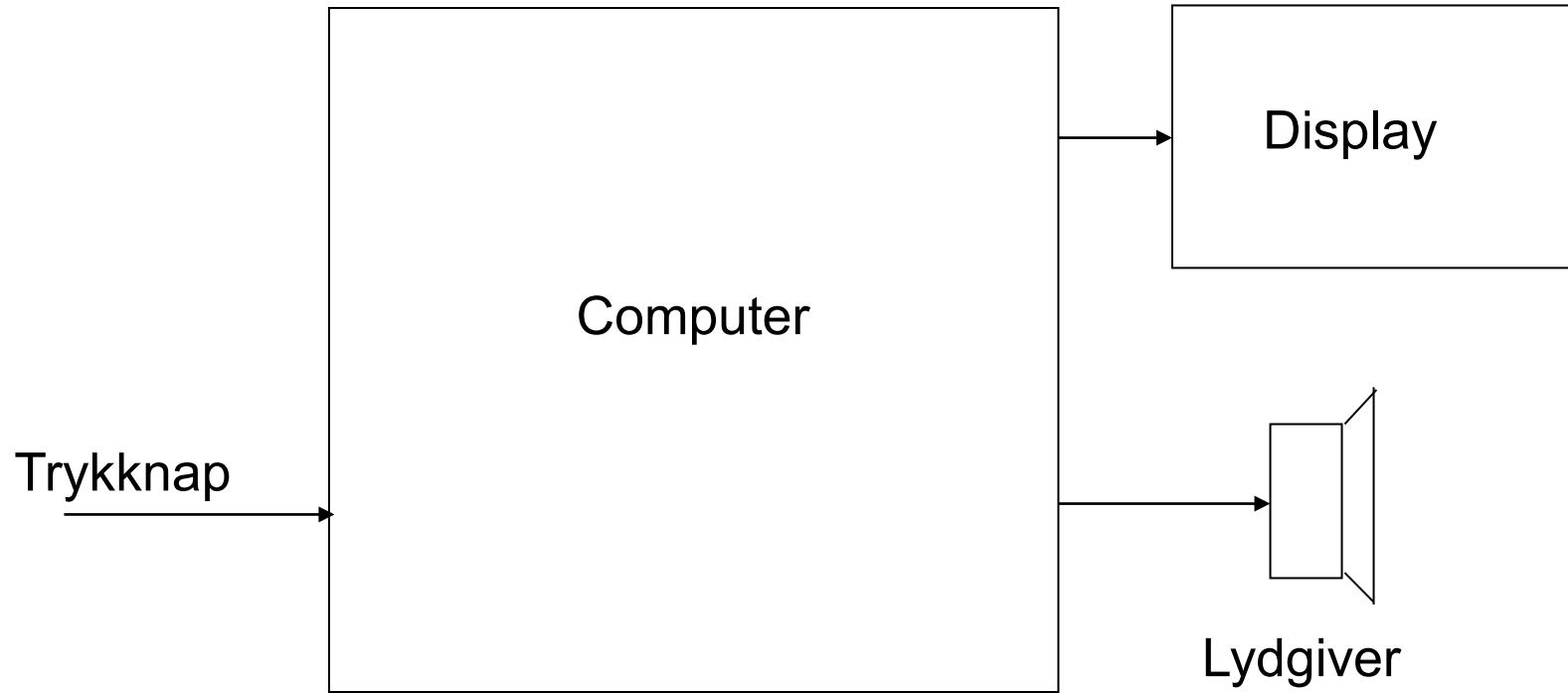


Slides from Henning Hargaard

Interrupt = (Midlertidig) afbrydelse



Eksempel



- * Computeren beregner hele tiden "noget", og viser det på displayet.
- * Hvis man trykker på trykknappen, skal lydgiveren sige "bib".

Løsning uden brug af interrupt

```
int main()
{
    [Klargør al hardware]
    while(1)
    {
        [Beregn noget]
        [Vis det på displayet]
        if ([Der er trykket på knappen])
            [Sig bib med lydgiveren]
    }
}
```

Hvad er ulempen?

Hvad er interrupts ?

- Et interrupt er en midlertidig afbrydelse af den igangværende programafvikling.
- Typisk startes et interrupt via et hardware-signal (en hændelse).
- Under interrupt'et afvikles en speciel funktion, der kaldes en interrupt-rutine (eller interrupt service routine = ISR). Denne er skrevet af programmøren.
- Efter interrupt'et vendes tilbage til den afbrudte programafvikling.

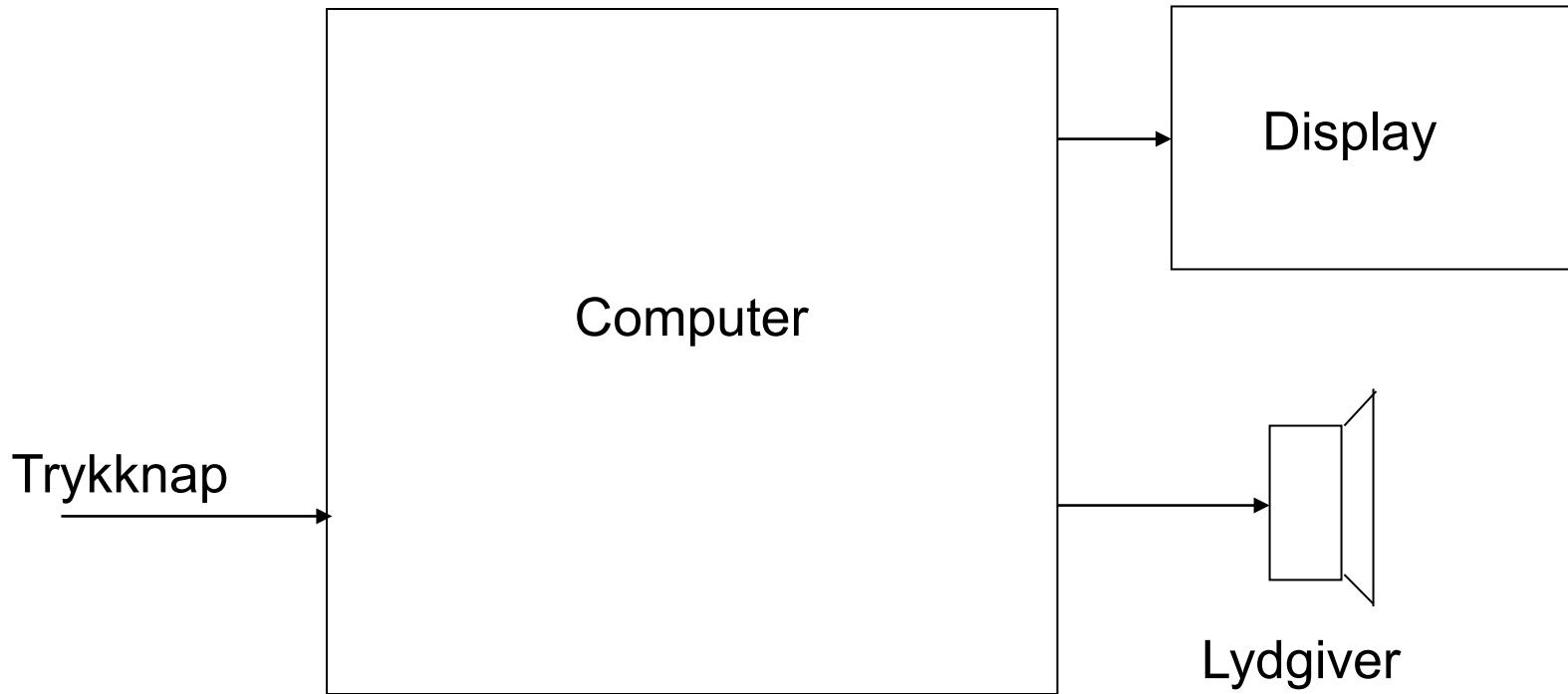
Løsning med brug af interrupt

```
// Denne funktion kaldes AUTOMATISK af HARDWARE  
// hver gang, der trykkes på knappen  
ISR (INT0_vect)  
{  
    [Sig bib med lydgiveren]  
}
```

```
int main()  
{  
    [Klargør al hardware]  
    [Klargør (enable) interrupt]  
    while(1)  
    {  
        [Beregn noget]  
        [Vis det på displayet]  
    }  
}
```

Fordele / ulemper?

Hvornår bør man overveje interrupts ?

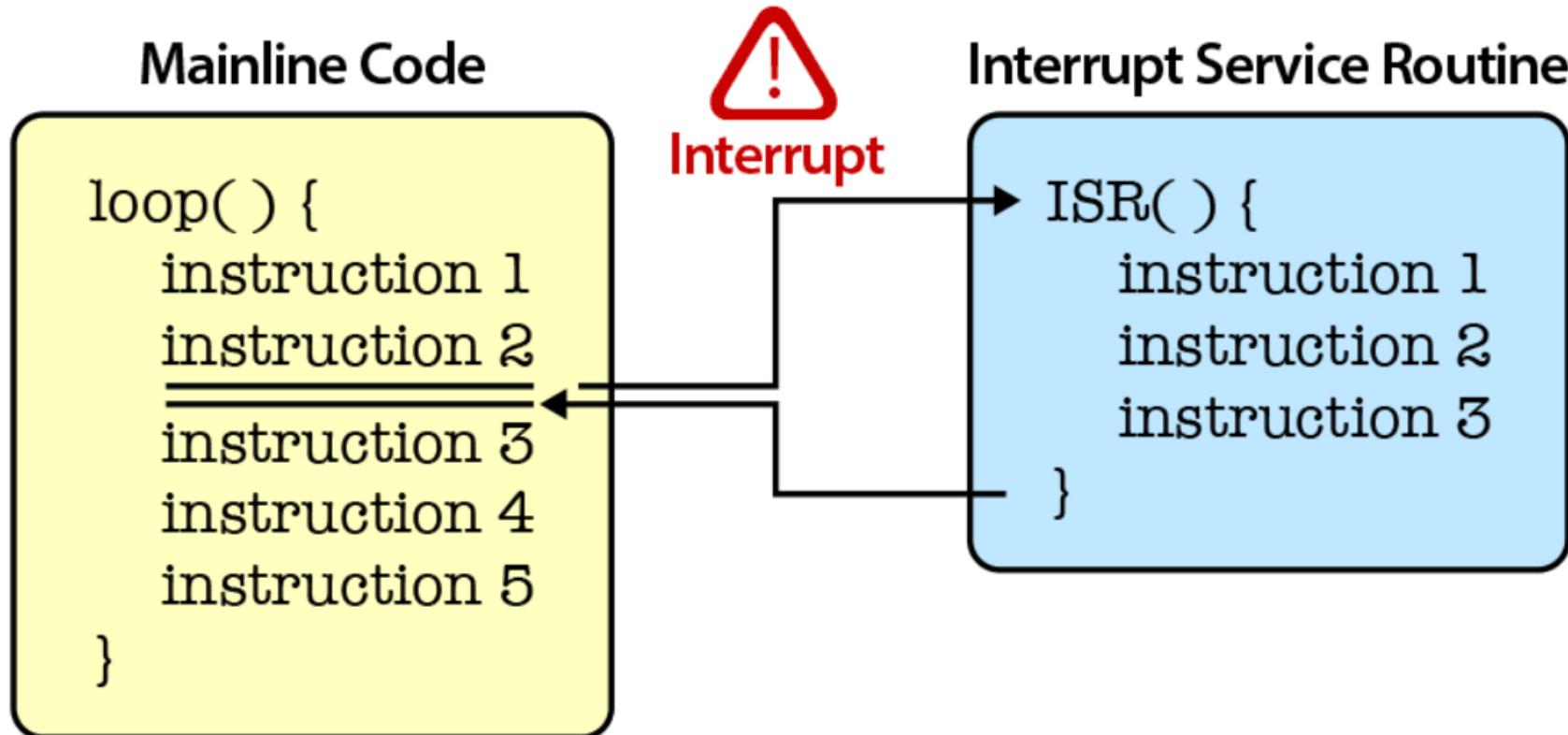


* Når der er hændelser, som sker på uforudsigelige tidspunkter.

og/eller

* Når man skal reagere hurtigt på hændelsen.

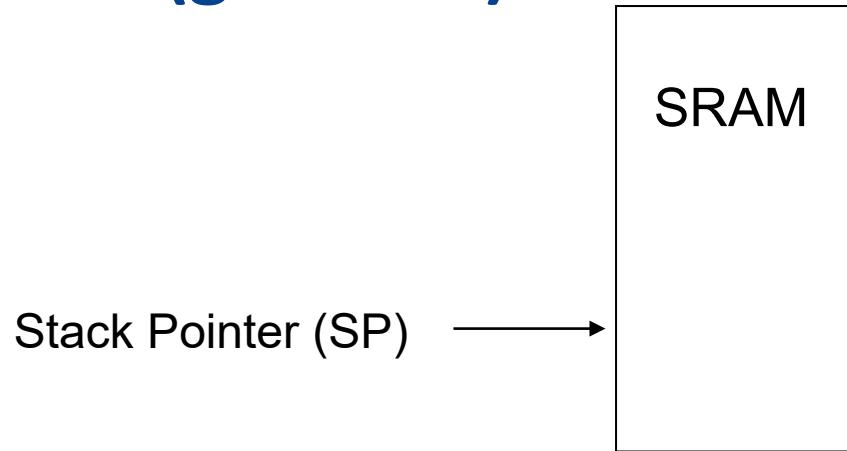
Interrupt - sekvensen



Interrupt - sekvensen

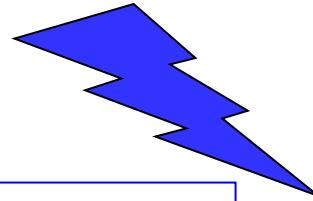
1. Igangværende instruktion færdiggøres.
2. Programtælleren gemmes på stacken.
(varer 4 clk-cykles)
3. Jump til interrupt-rutinen.
(varer 3 clock-perioder)
4. Global interrupt enable flag nulstilles.
Hvis "sei" i interrupt-rutinen => mulighed for "nested interrupts".
5. RETI => Programtælleren hentes fra stacken og global interrupt enable flag sættes igen.
(varer i alt 4 clk-perioder).

Hardware Stack (generelt)



- **Stack pointer'en** er et vigtigt register, som holder styr på, hvor "toppen af stacken" er (d.v.s. **næste ledige plads (adresse)**).
- Hver gang data skrives til stacken, flyttes SP **automatisk tilbage** (dekrementeres).
- Hver gang data hentes fra stacken, flyttes SP **automatisk frem** (inkrementeres).

Interrupts og stacken



Ved HW-interrupt
gemmes adressen
på næste instruktion
automatisk på
stacken !
SP dekrementeres.

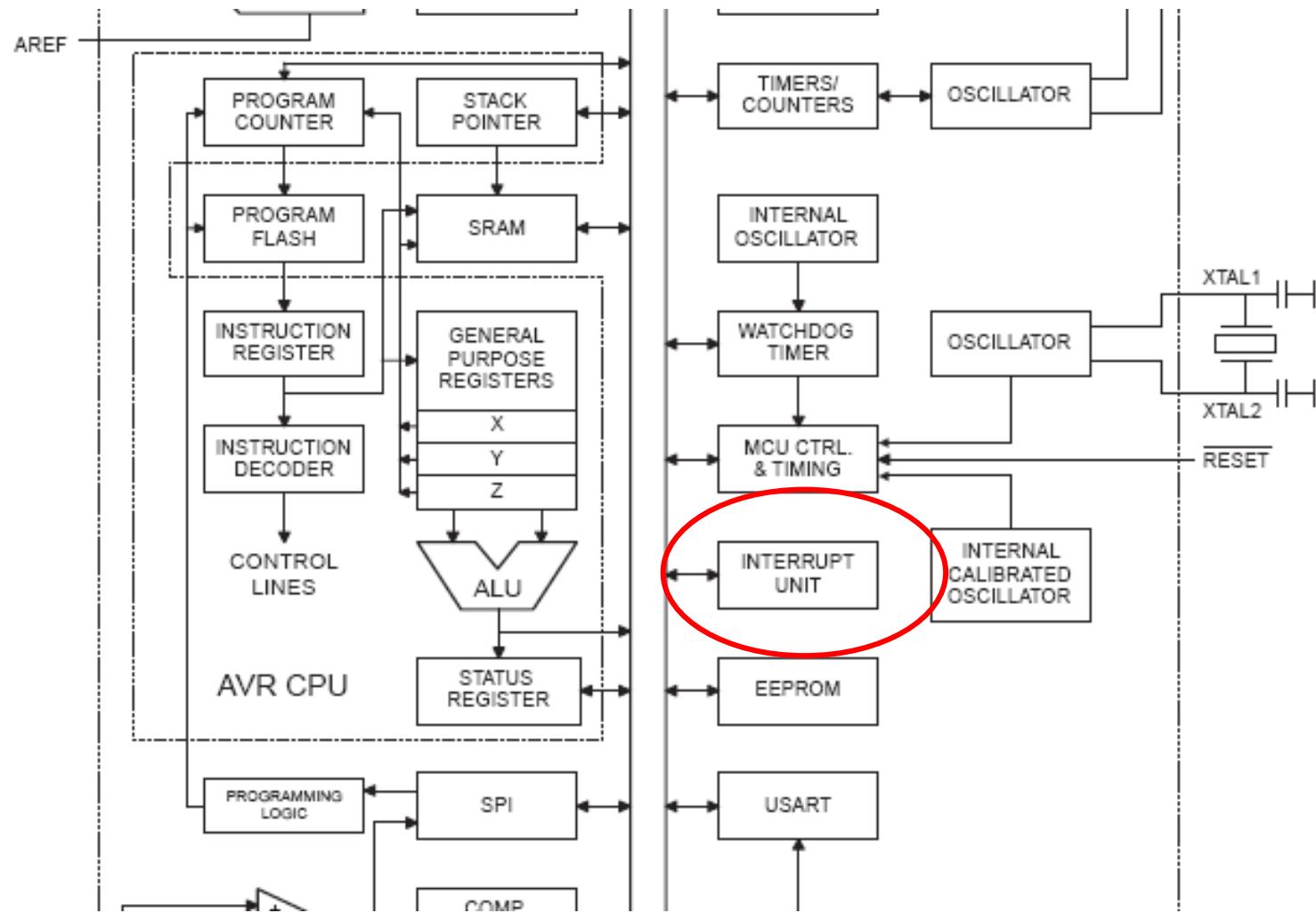
Interrupt:

```
MOV R0, R4  
INC R7  
MOV R1, R3  
RETI
```

Programtælleren (PC)
loades fra stacken.
SP inkrementeres.

- Ved interrupts anvendes stacken automatisk, så der fortsættes i programmet, hvor man blev afbrudt.

AVR interrupt unit



(Mega32: Interrupt vektorer)

Table 10-1: Interrupt Vector Table for the Mega32

Interrupt	ROM Location (Hex)
Reset	0000
External Interrupt request 0	0002
External Interrupt request 1	0004
External Interrupt request 2	0006
Time/Counter2 Compare Match	0008
Time/Counter2 Overflow	000A
Time/Counter1 Capture Event	000C
Time/Counter1 Compare Match A	000E
Time/Counter1 Compare Match B	0010
Time/Counter1 Overflow	0012
Time/Counter0 Compare Match	0014
Time/Counter0 Overflow	0016
SPI Transfer complete	0018
USART, Receive complete	001A
USART, Data Register Empty	001C
USART, Transmit Complete	001E
ADC Conversion complete	0020
EEPROM ready	0022
Analog Comparator	0024
Two-wire Serial Interface	0026
Store Program Memory Ready	0028

Mega2560: Interrupt vektorer (1 af 2)

Vector No.	Program Address ⁽¹⁾	Source	Interrupt Definition
1	\$0000 ⁽¹⁾	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	\$0002	INT0	External Interrupt Request 0
3	\$0004	INT1	External Interrupt Request 1
4	\$0006	INT2	External Interrupt Request 2
5	\$0008	INT3	External Interrupt Request 3
6	\$000A	INT4	External Interrupt Request 4
7	\$000C	INT5	External Interrupt Request 5
8	\$000E	INT6	External Interrupt Request 6
9	\$0010	INT7	External Interrupt Request 7
10	\$0012	PCINT0	Pin Change Interrupt Request 0
11	\$0014	PCINT1	Pin Change Interrupt Request 1
12	\$0016 ⁽²⁾	PCINT2	Pin Change Interrupt Request 2
13	\$0018	WDT	Watchdog Time-out Interrupt
14	\$001A	TIMER2 COMPA	Timer/Counter2 Compare Match A
15	\$001C	TIMER2 COMPB	Timer/Counter2 Compare Match B
16	\$001E	TIMER2 OVF	Timer/Counter2 Overflow
17	\$0020	TIMER1 CAPT	Timer/Counter1 Capture Event
18	\$0022	TIMER1 COMPA	Timer/Counter1 Compare Match A
19	\$0024	TIMER1 COMPB	Timer/Counter1 Compare Match B
20	\$0026	TIMER1 COMPC	Timer/Counter1 Compare Match C
21	\$0028	TIMER1 OVF	Timer/Counter1 Overflow
22	\$002A	TIMER0 COMPA	Timer/Counter0 Compare Match A
23	\$002C	TIMER0 COMPB	Timer/Counter0 Compare match B
24	\$002E	TIMER0 OVF	Timer/Counter0 Overflow
25	\$0030	SPI, STC	SPI Serial Transfer Complete
26	\$0032	USART0 RX	USART0 Rx Complete
27	\$0034	USART0 UDRE	USART0 Data Register Empty
28	\$0036	USART0 TX	USART0 Tx Complete
29	\$0038	ANALOG COMP	Analog Comparator
30	\$003A	ADC	ADC Conversion Complete

Mega2560: Interrupt vektorer (2 af 2)

Vector No.	Program Address ⁽²⁾	Source	Interrupt Definition
31	\$003C	EE READY	EEPROM Ready
32	\$003E	TIMER3 CAPT	Timer/Counter3 Capture Event
33	\$0040	TIMER3 COMPA	Timer/Counter3 Compare Match A
34	\$0042	TIMER3 COMPB	Timer/Counter3 Compare Match B
35	\$0044	TIMER3 COMPC	Timer/Counter3 Compare Match C
36	\$0046	TIMER3 OVF	Timer/Counter3 Overflow
37	\$0048	USART1 RX	USART1 Rx Complete
38	\$004A	USART1 UDRE	USART1 Data Register Empty
39	\$004C	USART1 TX	USART1 Tx Complete
40	\$004E	TWI	2-wire Serial Interface
41	\$0050	SPM READY	Store Program Memory Ready
42	\$0052 ⁽³⁾	TIMER4 CAPT	Timer/Counter4 Capture Event
43	\$0054	TIMER4 COMPA	Timer/Counter4 Compare Match A
44	\$0056	TIMER4 COMPB	Timer/Counter4 Compare Match B
45	\$0058	TIMER4 COMPC	Timer/Counter4 Compare Match C
46	\$005A	TIMER4 OVF	Timer/Counter4 Overflow
47	\$005C ⁽³⁾	TIMER5 CAPT	Timer/Counter5 Capture Event
48	\$005E	TIMER5 COMPA	Timer/Counter5 Compare Match A
49	\$0060	TIMER5 COMPB	Timer/Counter5 Compare Match B
50	\$0062	TIMER5 COMPC	Timer/Counter5 Compare Match C
51	\$0064	TIMER5 OVF	Timer/Counter5 Overflow
52	\$0066 ⁽³⁾	USART2 RX	USART2 Rx Complete
53	\$0068 ⁽³⁾	USART2 UDRE	USART2 Data Register Empty
54	\$006A ⁽³⁾	USART2 TX	USART2 Tx Complete
55	\$006C ⁽³⁾	USART3 RX	USART3 Rx Complete
56	\$006E ⁽³⁾	USART3 UDRE	USART3 Data Register Empty
57	\$0070 ⁽³⁾	USART3 TX	USART3 Tx Complete

Global interrupt enable

- Via et enkelt bit (Global Interrupt Enable Flag) **tillades/forbydes alle interrupts.**
Kan betragtes som interruptsystems **hovedafbryder**.
- Assembly-instruktionen **"sei"** tillader (enabler) interrupts.
- Assembly-instruktionen **"cli"** slukker (disabler) interrupts.

Kode – eksempler (AVR GCC)

```
// Global interrupt enable  
sei();
```

```
// Global interrupt disable  
cli();
```

Kode-eksempel (ISR)

```
#include <avr/interrupt.h>
ISR(INT1_vect) //ISR navn (= "type")
{
    //Her skrives koden
}
```

OBS: Må ikke skrives inden i main() – eller inden i en anden funktion.

AVR GCC ISR typenavne (Mega32)

Table 10-3: Interrupt Vector Name for the ATmega32/ATmega16 in WinAVR

Interrupt	Vector Name in WinAVR
External Interrupt request 0	INT0_vect
External Interrupt request 1	INT1_vect
External Interrupt request 2	INT2_vect
Time/Counter2 Compare Match	TIMER2_COMP_vect
Time/Counter2 Overflow	TIMER2_OVF_vect
Time/Counter1 Capture Event	TIMER1_CAPT_vect
Time/Counter1 Compare Match A	TIMER1_COMPA_vect
Time/Counter1 Compare Match B	TIMER1_COMPB_vect
Time/Counter1 Overflow	TIMER1_OVF_vect
Time/Counter0 Compare Match	TIMER0_COMP_vect
Time/Counter0 Overflow	TIMER0_OVF_vect
SPI Transfer complete	SPI_STC_vect
USART, Receive complete	USART0_RX_vect
USART, Data Register Empty	USART0_UDRE_vect
USART, Transmit Complete	USART0_TX_vect
ADC Conversion complete	ADC_vect
EEPROM ready	EE_RDY_vect
Analog Comparator	ANA_COMP_vect
Two-wire Serial Interface	TWI_vect
Store Program Memory Ready	SPM_RDY_vect

Mega2560 ISR typenavne (Pins)

External interrupts	ISR name
INT0	INT0_vect
INT1	INT1_vect
INT2	INT2_vect
INT3	INT3_vect
INT4	INT4_vect
INT5	INT5_vect
INT6	INT6_vect
INT7	INT7_vect

PIN change interrupts	ISR name
PCINT0	PCINT0_vect
PCINT1	PCINT1_vect
PCINT2	PCINT2_vect

Mega2560 ISR typenavne (Timers)

Timer 0 interrupts	ISR name	Timer 3 interrupts	ISR name
COMP A	TIMER0_COMPA_vect	Capture	TIMER3_CAPT_vect
COMP B	TIMER0_COMPB_vect	COMP A	TIMER3_COMPA_vect
Overflow	TIMER0_OVF_vect	COMP B	TIMER3_COMPB_vect
Timer 1 interrupts	ISR name	COMP C	TIMER3_COMPC_vect
Capture	TIMER1_CAPT_vect	Overflow	TIMER3_OVF_vect
COMP A	TIMER1_COMPA_vect	Timer 4 interrupts	ISR name
COMP B	TIMER1_COMPB_vect	Capture	TIMER4_CAPT_vect
COMP C	TIMER1_COMPC_vect	COMP A	TIMER4_CONA_vect
Overflow	TIMER1_OVF_vect	COMP B	TIMER4_CONB_vect
Timer 2 interrupts	ISR name	COMP C	TIMER4_CONC_vect
COMP A	TIMER2_COMPA_vect	Overflow	TIMER4_OVF_vect
COMP B	TIMER2_COMPB_vect		
Overflow	TIMER2_OVF_vect		
Timer 3 interrupts	ISR name		
Capture	TIMER3_CAPT_vect		
Timer 4 interrupts	ISR name	Timer 5 interrupts	ISR name
COMP A	TIMER4_COMPA_vect	Capture	TIMER5_CAPT_vect
COMP B	TIMER4_COMPB_vect	COMP A	TIMER5_COMPA_vect
COMP C	TIMER4_COMPC_vect	COMP B	TIMER5_COMPB_vect
Overflow	TIMER4_OVF_vect	COMP C	TIMER5_COMPC_vect
		Overflow	TIMER5_OVF_vect

Mega2560 ISR typenavne (USART)

USART 0 interrupts	ISR name
RX	USART0_RX_vect
TX	USART0_TX_vect
UDR Empty	USART0_UDRE_vect

USART 2 interrupts	ISR name
RX	USART2_RX_vect
TX	USART2_TX_vect
UDR Empty	USART2_UDRE_vect

USART 1 interrupts	ISR name
RX	USART1_RX_vect
TX	USART1_TX_vect
UDR Empty	USART1_UDRE_vect

USART 3 interrupts	ISR name
RX	USART3_RX_vect
TX	USART3_TX_vect
UDR Empty	USART3_UDRE_vect

Mega2560 ISR typenavne (diverse)

Other interrupts	ISRname
Watchdog timeout	WDT_vect
SPI	SPI_STC_vect
Analog comparator	ANALOG_COMP_vect
ADC	ADC_vect
EEPROM	EE_READY_vect
TWI	TWI_vect
SPM ready	SPM_READY_vect

Lokale interrupt enables

- De enkelte interrupttyper har hver især et **lokalt** interrupt enable flag.
- **Kun** hvis dette lokale interrupt enable flag er sat, **OG** det globale interrupt flag er sat, vil enheden kunne starte et interrupt !

(Mega32: Eksterne interrupts)

Table 10-1: Interrupt Vector Table for the Mega32

Interrupt	ROM Location (Hex)
Reset	0000
External Interrupt request 0	0002
External Interrupt request 1	0004
External Interrupt request 2	0006
Time/Counter2 Compare Match	0008
Time/Counter2 Overflow	000A
Time/Counter1 Capture Event	000C
Time/Counter1 Compare Match A	000E
Time/Counter1 Compare Match B	0010
Time/Counter1 Overflow	0012
Time/Counter0 Compare Match	0014
Time/Counter0 Overflow	0016
SPI Transfer complete	0018
USART, Receive complete	001A
USART, Data Register Empty	001C
USART, Transmit Complete	001E
ADC Conversion complete	0020
EEPROM ready	0022
Analog Comparator	0024
Two-wire Serial Interface	0026
Store Program Memory Ready	0028

(Mega32: Eksterne interrupt ben)

(XCK/T0) PB0	1	40	PA0 (ADC0)
(T1) PB1	2	39	PA1 (ADC1)
(INT2/AIN0) PB2	3	38	PA2 (ADC2)
(OC0/AIN1) PB3	4	37	PA3 (ADC3)
(SS) PB4	5	36	PA4 (ADC4)
(MOSI) PB5	6	35	PA5 (ADC5)
(MISO) PB6	7	34	PA6 (ADC6)
(SCK) PB7	8	33	PA7 (ADC7)
RESET	9	32	AREF
VCC	10	31	GND
GND	11	30	AVCC
XTAL2	12	29	PC7 (TOSC2)
XTAL1	13	28	PC6 (TOSC1)
(RXD) PD0	14	27	PC5 (TDI)
(TXD) PD1	15	26	PC4 (TDO)
(INT0) PD2	16	25	PC3 (TMS)
(INT1) PD3	17	24	PC2 (TCK)
(OC1B) PD4	18	23	PC1 (SDA)
(OC1A) PD5	19	22	PC0 (SCL)
(ICP1) PD6	20	21	PD7 (OC2)

Mega2560: Eksterne interrupts

Vector No.	Program Address ⁽²⁾	Source	Interrupt Definition
1	\$0000 ⁽¹⁾	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	\$0002	INT0	External Interrupt Request 0
3	\$0004	INT1	External Interrupt Request 1
4	\$0006	INT2	External Interrupt Request 2
5	\$0008	INT3	External Interrupt Request 3
6	\$000A	INT4	External Interrupt Request 4
7	\$000C	INT5	External Interrupt Request 5
8	\$000E	INT6	External Interrupt Request 6
9	\$0010	INT7	External Interrupt Request 7
10	\$0012	PCINT0	Pin Change Interrupt Request 0
11	\$0014	PCINT1	Pin Change Interrupt Request 1
12	\$0016 ⁽³⁾	PCINT2	Pin Change Interrupt Request 2
13	\$0018	WDT	Watchdog Time-out Interrupt
14	\$001A	TIMER2 COMPA	Timer/Counter2 Compare Match A

Mega2560: Eksterne interrupt ben

INT0 = PD, ben 0

INT1 = PD, ben 1

INT2 = PD, ben 2 = SW2 på Mega2560 I/O shield

INT3 = PD, ben 3 = SW3 på Mega2560 I/O shield

INT4 = PE, ben 4

INT5 = PE, ben 5

INT6 = PE, ben 6

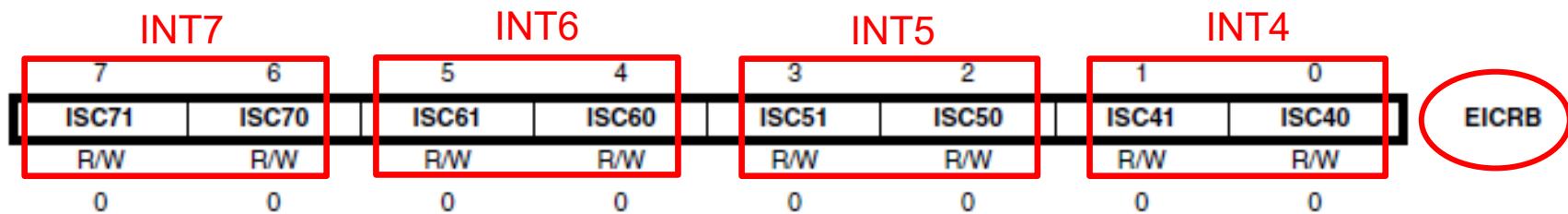
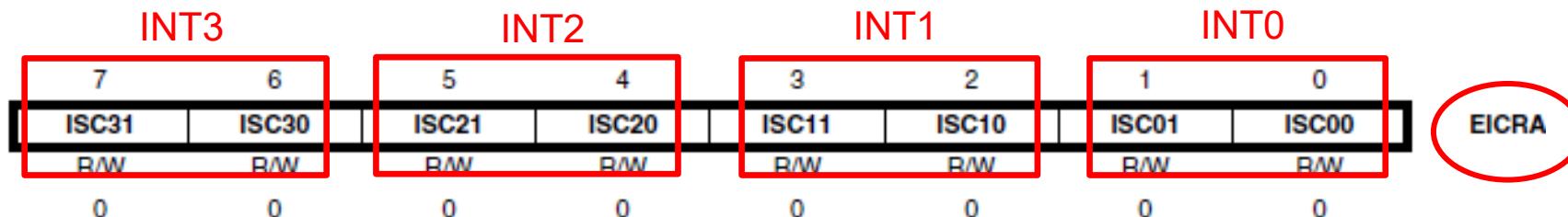
INT7 = PE, ben 7

Mega2560: Ekstern interrupt enable bits

7	6	5	4	3	2	1	0	EIMSK
INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0	
R/W								

- **0** : Disabled.
- **1** : Enabled.
- Kodeeksempel :
 $EIMSK = EIMSK \mid 0b00001100;$
-- eller --
 $EIMSK |= 0b00001100;$

Mega2560: Trignings-muligheder, INT0-INT7



ISCn1	ISCn0	Description
0	0	The low level of INTn generates an interrupt request
0	1	Any edge of INTn generates asynchronously an interrupt request
1	0	The falling edge of INTn generates asynchronously an interrupt request
1	1	The rising edge of INTn generates asynchronously an interrupt request

(Mega2560: Eksterne interrupts: Flags)

7	6	5	4	3	2	1	0
INTF7	INTF6	INTF5	INTF4	INTF3	INTF2	INTF1	IINTF0
R/W							
0	0	0	0	0	0	0	0

EIFR

- Sættes, når interrupt triges (FLIP-FLOP).
- Hvis globalt interrupt enabled og lokalt enabled:
Interrupt starter herefter.
- Flag nulstilles automatisk i interrupt rutinen
(eller af SW ved at skrive 1 til det aktuelle bit).
- Det er sjældent, at vi har behov for at bruge dette register (når interrupt er enabled).

(Mega2560: Pin change interrupts (PCI))

- Mega2560 har desuden mulighed for noget, der kaldes **“Pin Change Interrupts”** (der har Mega32 ikke).
- Princippet er, at vi vil kunne få **interrupt**, hvis et (ud af i alt 24) ben skifter tilstand (low->high eller high->low).
- PCI **behandles ikke i dette kursus**, men er naturligvis beskrevet i Mega2560 databogen (findes på MSYS Brightspace).

Test ("socrative.com"): Room = AUQUIZ

Vi ønsker at enable Mega2560's eksterne interrupts INT0 og INT4 (de øvrige interrupts skal være disabled). INT0 skal trigge på "falling edge" og INT4 skal trigge på "rising edge". Hvilken kode er korrekt initiering ?

- A: EICRA = 0b00000010;
EICRB = 0b00000011;
EIMSK = 0b00010001;
- B: EICRA = 0b00000011;
EICRB = 0b00000010;
EIMSK = 0b00010001;
- C: EICRA = 0b00000010;
EICRB = 0b00000000;
EIMSK = 0b10000001;



Test ("socrative.com"): Room = AUQUIZ

For Mega2560 er det eksterne interrupt INT0 initieret på følgende måde:

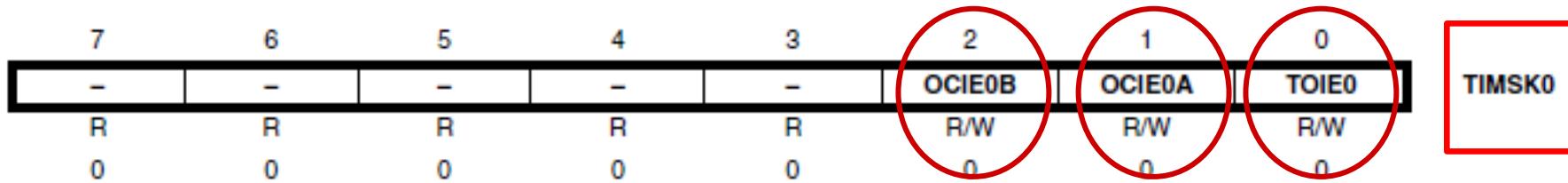
```
EICRA = 0b00000001;  
EIMSK |= 0b00000001;  
sei( );
```



Hvilken hændelse på INT0 benet vil starte et interrupt ?

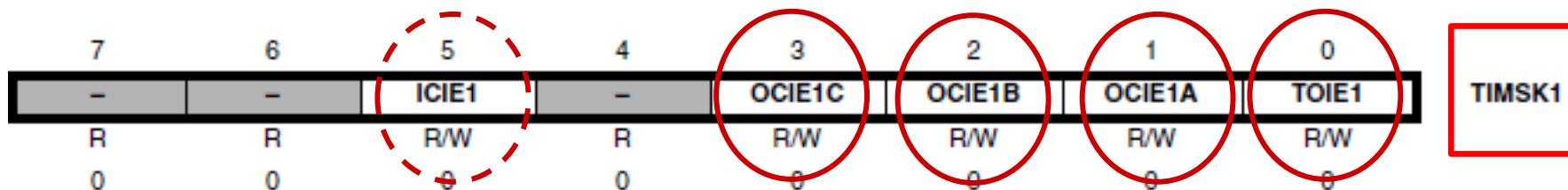
- A: Når signalet et lavt (0 volt).
- B: Når signalet ændrer tilstand ("any logical change").
- C: Når signalet går fra høj til lav ("falling edge").
- D: Når signalet går fra lav til høj ("rising edge");

Mega2560: Timer 0 interrupt enables



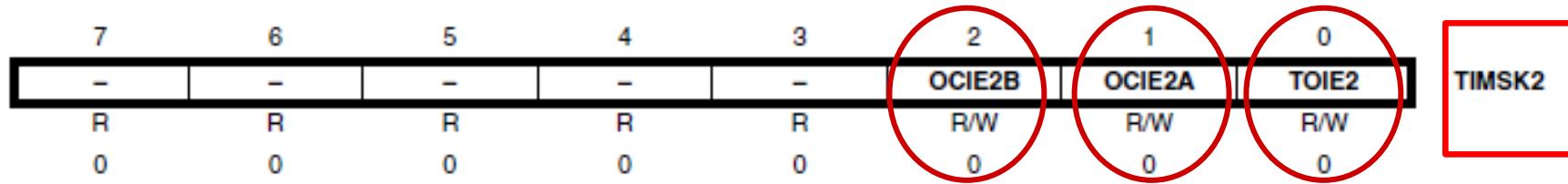
- Bit **TOIE0** (bit 0):
0 => **Disable** overflow interrupt.
1 => **Enable** overflow interrupt.
- Bit **OCIE0A** (bit 1):
0 => **Disable** output compare interrupt, A-systemet.
1 => **Enable** output compare interrupt, A-systemet.
- Bit **OCIE0B** (bit 2):
0 => **Disable** output compare interrupt, B-systemet.
1 => **Enable** output compare interrupt, B-systemet.

Mega2560: Timer 1 interrupt enables



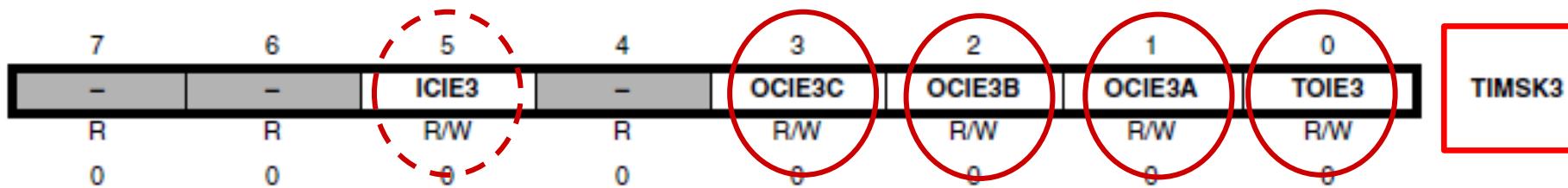
- Bit **TOIE1** (bit 0):
0 => **Disable** overflow interrupt.
1 => **Enable** overflow interrupt.
- Bit **OCIE1A** (bit 1):
0 => **Disable** output compare interrupt, A-systemet.
1 => **Enable** output compare interrupt, A-systemet.
- Bit **OCIE1B** (bit 2):
0 => **Disable** output compare interrupt, B-systemet.
1 => **Enable** output compare interrupt, B-systemet.
- Bit **OCIE1C** (bit 3):
0 => **Disable** output compare interrupt, C-systemet.
1 => **Enable** output compare interrupt, C-systemet.

Mega2560: Timer 2 interrupt enables



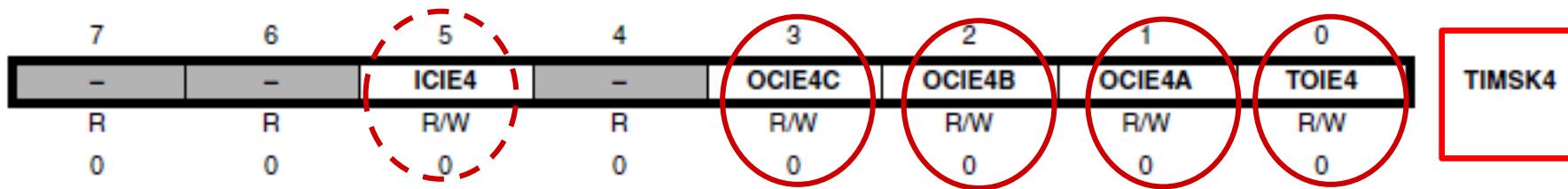
- Bit **TOIE2** (bit 0):
0 => **Disable** overflow interrupt.
1 => **Enable** overflow interrupt.
- Bit **OCIE2A** (bit 1):
0 => **Disable** output compare interrupt, A-systemet.
1 => **Enable** output compare interrupt, A-systemet.
- Bit **OCIE2B** (bit 2):
0 => **Disable** output compare interrupt, B-systemet.
1 => **Enable** output compare interrupt, B-systemet.

Mega2560: Timer 3 interrupt enables



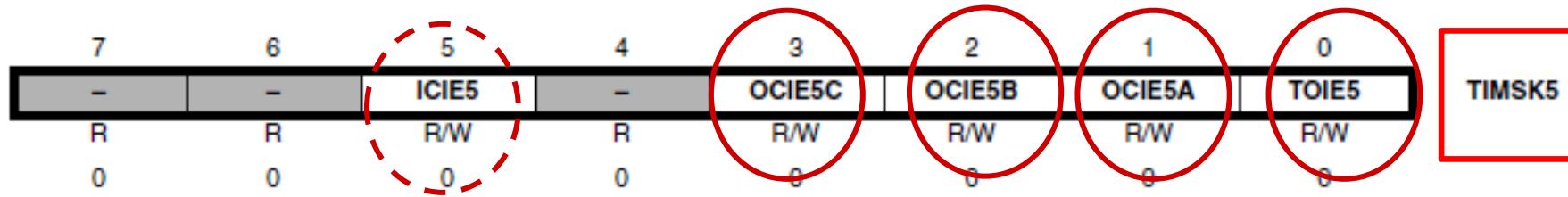
- Bit **TOIE3** (bit 0):
0 => **Disable** overflow interrupt.
1 => **Enable** overflow interrupt.
- Bit **OCIE3A** (bit 1):
0 => **Disable** output compare interrupt, A-systemet.
1 => **Enable** output compare interrupt, A-systemet.
- Bit **OCIE3B** (bit 2):
0 => **Disable** output compare interrupt, B-systemet.
1 => **Enable** output compare interrupt, B-systemet.
- Bit **OCIE3C** (bit 3):
0 => **Disable** output compare interrupt, C-systemet.
1 => **Enable** output compare interrupt, C-systemet.

Mega2560: Timer 4 interrupt enables



- Bit **TOIE4** (bit 0):
0 => **Disable** overflow interrupt.
1 => **Enable** overflow interrupt.
- Bit **OCIE4A** (bit 1):
0 => **Disable** output compare interrupt, A-systemet.
1 => **Enable** output compare interrupt, A-systemet.
- Bit **OCIE4B** (bit 2):
0 => **Disable** output compare interrupt, B-systemet.
1 => **Enable** output compare interrupt, B-systemet.
- Bit **OCIE4C** (bit 3):
0 => **Disable** output compare interrupt, C-systemet.
1 => **Enable** output compare interrupt, C-systemet.

Mega2560: Timer 5 interrupt enables



- Bit **TOIE5** (bit 0):
0 => **Disable** overflow interrupt.
1 => **Enable** overflow interrupt.
- Bit **OCIE5A** (bit 1):
0 => **Disable** output compare interrupt, A-systemet.
1 => **Enable** output compare interrupt, A-systemet.
- Bit **OCIE5B** (bit 2):
0 => **Disable** output compare interrupt, B-systemet.
1 => **Enable** output compare interrupt, B-systemet.
- Bit **OCIE5C** (bit 3):
0 => **Disable** output compare interrupt, C-systemet.
1 => **Enable** output compare interrupt, C-systemet.

Mega2560: Timer 0, 1 og 2 interrupts

14	\$001A	TIMER2 COMPA	Timer/Counter2 Compare Match A
15	\$001C	TIMER2 COMPB	Timer/Counter2 Compare Match B
16	\$001E	TIMER2 OVF	Timer/Counter2 Overflow
17	\$0020	TIMER1 CAPT	Timer/Counter1 Capture Event
18	\$0022	TIMER1 COMPA	Timer/Counter1 Compare Match A
19	\$0024	TIMER1 COMPB	Timer/Counter1 Compare Match B
20	\$0026	TIMER1 COMPC	Timer/Counter1 Compare Match C
21	\$0028	TIMER1 OVF	Timer/Counter1 Overflow
22	\$002A	TIMER0 COMPA	Timer/Counter0 Compare Match A
23	\$002C	TIMER0 COMPB	Timer/Counter0 Compare match B
24	\$002E	TIMER0 OVF	Timer/Counter0 Overflow

Mega2560: Timer 3, 4 og 5 interrupts

32	\$003E	TIMER3 CAPT	Timer/Counter3 Capture Event
33	\$0040	TIMER3 COMPA	Timer/Counter3 Compare Match A
34	\$0042	TIMER3 COMPB	Timer/Counter3 Compare Match B
35	\$0044	TIMER3 COMPC	Timer/Counter3 Compare Match C
36	\$0046	TIMER3 OVF	Timer/Counter3 Overflow
37	\$0048	USART1 RX	USART1 Rx Complete
38	\$004A	USART1 UDRE	USART1 Data Register Empty
39	\$004C	USART1 TX	USART1 Tx Complete
40	\$004E	TWI	2-wire Serial Interface
41	\$0050	SPM READY	Store Program Memory Ready
42	\$0052 ⁽³⁾	TIMER4 CAPT	Timer/Counter4 Capture Event
43	\$0054	TIMER4 COMPA	Timer/Counter4 Compare Match A
44	\$0056	TIMER4 COMPB	Timer/Counter4 Compare Match B
45	\$0058	TIMER4 COMPC	Timer/Counter4 Compare Match C
46	\$005A	TIMER4 OVF	Timer/Counter4 Overflow
47	\$005C ⁽³⁾	TIMER5 CAPT	Timer/Counter5 Capture Event
48	\$005E	TIMER5 COMPA	Timer/Counter5 Compare Match A
49	\$0060	TIMER5 COMPB	Timer/Counter5 Compare Match B
50	\$0062	TIMER5 COMPC	Timer/Counter5 Compare Match C
51	\$0064	TIMER5 OVF	Timer/Counter5 Overflow

Test ("socrative.com"): Room = AUQUIZ

Hvordan enables Timer 1 overflow interrupts for Mega2560 ?

- A: `TIMSK |= 0b00000001;`
`sei();`
- B: `TIMSK1 |= 0b00000001;`
`sei();`
- C: `TIMSK1 |= 0b01000000;`
`sei();`
- D: `TIMSK &= 0b00000001;`
`sei();`

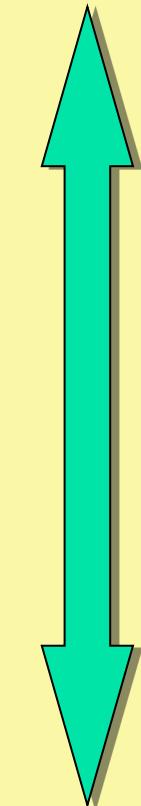


(Mega32: Interrupt prioritering)

Table 10-1: Interrupt Vector Table for the Mega32

Interrupt	ROM Location (Hex)
Reset	0000
External Interrupt request 0	0002
External Interrupt request 1	0004
External Interrupt request 2	0006
Time/Counter2 Compare Match	0008
Time/Counter2 Overflow	000A
Time/Counter1 Capture Event	000C
Time/Counter1 Compare Match A	000E
Time/Counter1 Compare Match B	0010
Time/Counter1 Overflow	0012
Time/Counter0 Compare Match	0014
Time/Counter0 Overflow	0016
SPI Transfer complete	0018
USART, Receive complete	001A
USART, Data Register Empty	001C
USART, Transmit Complete	001E
ADC Conversion complete	0020
EEPROM ready	0022
Analog Comparator	0024
Two-wire Serial Interface	0026
Store Program Memory Ready	0028

Highest priority

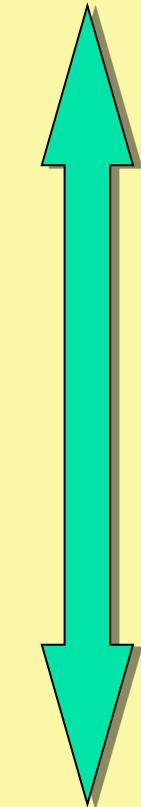


Lowest priority

Mega2560: Interrupt prioritering (1 af 2)

Vector No.	Program Address ⁽²⁾	Source	Interrupt Definition
1	\$0000 ⁽¹⁾	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	\$0002	INT0	External Interrupt Request 0
3	\$0004	INT1	External Interrupt Request 1
4	\$0006	INT2	External Interrupt Request 2
5	\$0008	INT3	External Interrupt Request 3
6	\$000A	INT4	External Interrupt Request 4
7	\$000C	INT5	External Interrupt Request 5
8	\$000E	INT6	External Interrupt Request 6
9	\$0010	INT7	External Interrupt Request 7
10	\$0012	PCINT0	Pin Change Interrupt Request 0
11	\$0014	PCINT1	Pin Change Interrupt Request 1
12	\$0016 ⁽³⁾	PCINT2	Pin Change Interrupt Request 2
13	\$0018	WDT	Watchdog Time-out Interrupt
14	\$001A	TIMER2 COMPA	Timer/Counter2 Compare Match A
15	\$001C	TIMER2 COMPB	Timer/Counter2 Compare Match B
16	\$001E	TIMER2 OVF	Timer/Counter2 Overflow
17	\$0020	TIMER1 CAPT	Timer/Counter1 Capture Event
18	\$0022	TIMER1 COMPA	Timer/Counter1 Compare Match A
19	\$0024	TIMER1 COMPB	Timer/Counter1 Compare Match B
20	\$0026	TIMER1 COMPC	Timer/Counter1 Compare Match C
21	\$0028	TIMER1 OVF	Timer/Counter1 Overflow
22	\$002A	TIMER0 COMPA	Timer/Counter0 Compare Match A
23	\$002C	TIMER0 COMPB	Timer/Counter0 Compare match B
24	\$002E	TIMER0 OVF	Timer/Counter0 Overflow
25	\$0030	SPI, STC	SPI Serial Transfer Complete
26	\$0032	USART0 RX	USART0 Rx Complete
27	\$0034	USART0 UDRE	USART0 Data Register Empty
28	\$0036	USART0 TX	USART0 Tx Complete
29	\$0038	ANALOG COMP	Analog Comparator
30	\$003A	ADC	ADC Conversion Complete

Highest priority

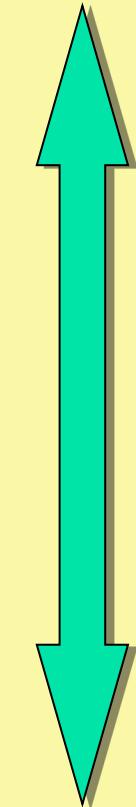


Lowest priority

Mega2560: Interrupt prioritering (2 af 2)

Vector No.	Program Address ⁽²⁾	Source	Interrupt Definition
31	\$003C	EE READY	EEPROM Ready
32	\$003E	TIMER3 CAPT	Timer/Counter3 Capture Event
33	\$0040	TIMER3 COMPA	Timer/Counter3 Compare Match A
34	\$0042	TIMER3 COMPB	Timer/Counter3 Compare Match B
35	\$0044	TIMER3 COMPC	Timer/Counter3 Compare Match C
36	\$0046	TIMER3 OVF	Timer/Counter3 Overflow
37	\$0048	USART1 RX	USART1 Rx Complete
38	\$004A	USART1 UDRE	USART1 Data Register Empty
39	\$004C	USART1 TX	USART1 Tx Complete
40	\$004E	TWI	2-wire Serial Interface
41	\$0050	SPM READY	Store Program Memory Ready
42	\$0052 ⁽³⁾	TIMER4 CAPT	Timer/Counter4 Capture Event
43	\$0054	TIMER4 COMPA	Timer/Counter4 Compare Match A
44	\$0056	TIMER4 COMPB	Timer/Counter4 Compare Match B
45	\$0058	TIMER4 COMPC	Timer/Counter4 Compare Match C
46	\$005A	TIMER4 OVF	Timer/Counter4 Overflow
47	\$005C ⁽³⁾	TIMER5 CAPT	Timer/Counter5 Capture Event
48	\$005E	TIMER5 COMPA	Timer/Counter5 Compare Match A
49	\$0060	TIMER5 COMPB	Timer/Counter5 Compare Match B
50	\$0062	TIMER5 COMPC	Timer/Counter5 Compare Match C
51	\$0064	TIMER5 OVF	Timer/Counter5 Overflow
52	\$0066 ⁽³⁾	USART2 RX	USART2 Rx Complete
53	\$0068 ⁽³⁾	USART2 UDRE	USART2 Data Register Empty
54	\$006A ⁽³⁾	USART2 TX	USART2 Tx Complete
55	\$006C ⁽³⁾	USART3 RX	USART3 Rx Complete
56	\$006E ⁽³⁾	USART3 UDRE	USART3 Data Register Empty
57	\$0070 ⁽³⁾	USART3 TX	USART3 Tx Complete

Highest priority



Lowest priority

Interrupt inden i et interrupt ("nesting")

- Det globale interrupt-flag nulstilles (automatisk), når man kommer ind I en ISR.
Derfor bliver **interrupts disabled**.
- Det globale interruptflag bliver (automatisk) sat, når **RETI** udføres.
Derfor bliver **interrupt enabled igen**.
- Hvis man vil tillade et interrupt i et andet interrupt, kan man – i starten af ISR'en – udføre `sei()`.

Interrupts og resource konflikt

■ Hvad kan gå galt i nedenstående?

```
1 .INCLUDE "M32DEF.INC"
2 .ORG 0x0      ;location for reset
3     JMP MAIN
4 .ORG 0x14    ;Timer0 compare match
5     JMP T0_CM_ISR
6 -----main program-----
7 .ORG 0x100
8 MAIN: LDI R20,HIGH(RAMEND)
9     OUT SPH,R20
10    LDI R20,LOW(RAMEND)
11    OUT SPL,R20 ;set up stack
12    SBI DDRB,5 ;PB5 = output
13    LDI R20,160
14    OUT OCR0,R20
15    LDI R20,0x09
16    OUT TCCR0,R20
17
18
19
20
21
22
23
24 HERE: OUT PORTC,R20
25     INC R20
26     JMP HERE
27 -----ISR for Timer0
28 T0_CM_ISR:
29     IN  R20,PIND
30     INC R20
31     OUT PORTD,R20
32     RETI
```

Løsning 1: Brug af forskellige registre

- Vælg nogen andre registre (bøvlet).

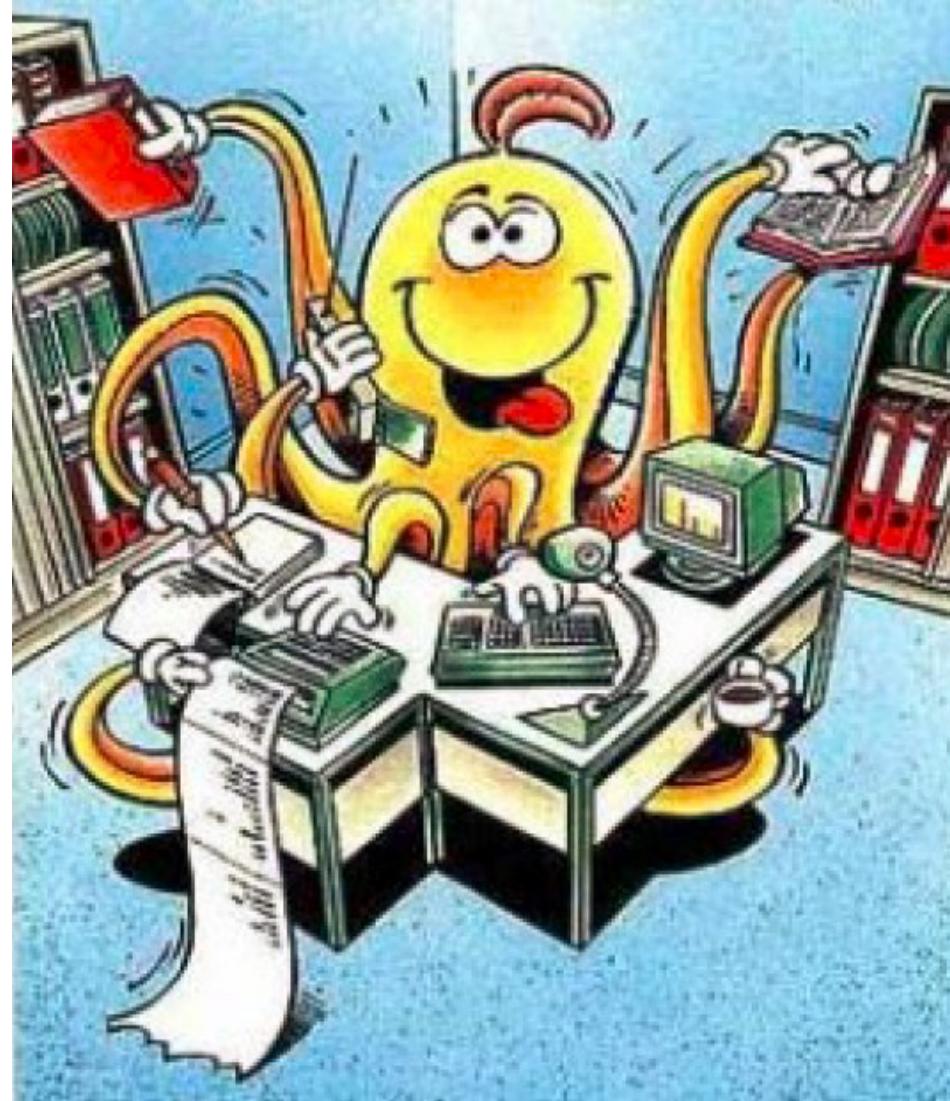
1 .INCLUDE "M32DEF.INC"	17 LDI R20,(1<<OCIE0)
2 .ORG 0x0 ;location for reset	18 OUT TIMSK,R20
3 JMP MAIN	19 SEI
4 .ORG 0x14 ;Timer0 compare match	20 LDI R20,0xFF
5 JMP T0_CM_ISR	21 OUT DDRC,R20
6 ;-----main program-----	22 OUT DDRD,R20
7 .ORG 0x100	23 LDI R20, 0
8 MAIN: LDI R20,HIGH(RAMEND)	24 HERE: OUT PORTC,R20
9 OUT SPH,R20	25 INC R20
10 LDI R20,LOW(RAMEND)	26 JMP HERE
11 OUT SPL,R20 ;set up stack	27 ;-----ISR for Timer0
12 SBI DDRB,5 ;PB5 = output	28 T0_CM_ISR:
13 LDI R20,160	29 IN R21,PIND
14 OUT OCR0,R20	30 INC R21
15 LDI R20,0x09	31 OUT PORTD,R21
16 OUT TCCR0,R20	32 RETI

Løsning 2: “Context saving” (PUSH/POP)

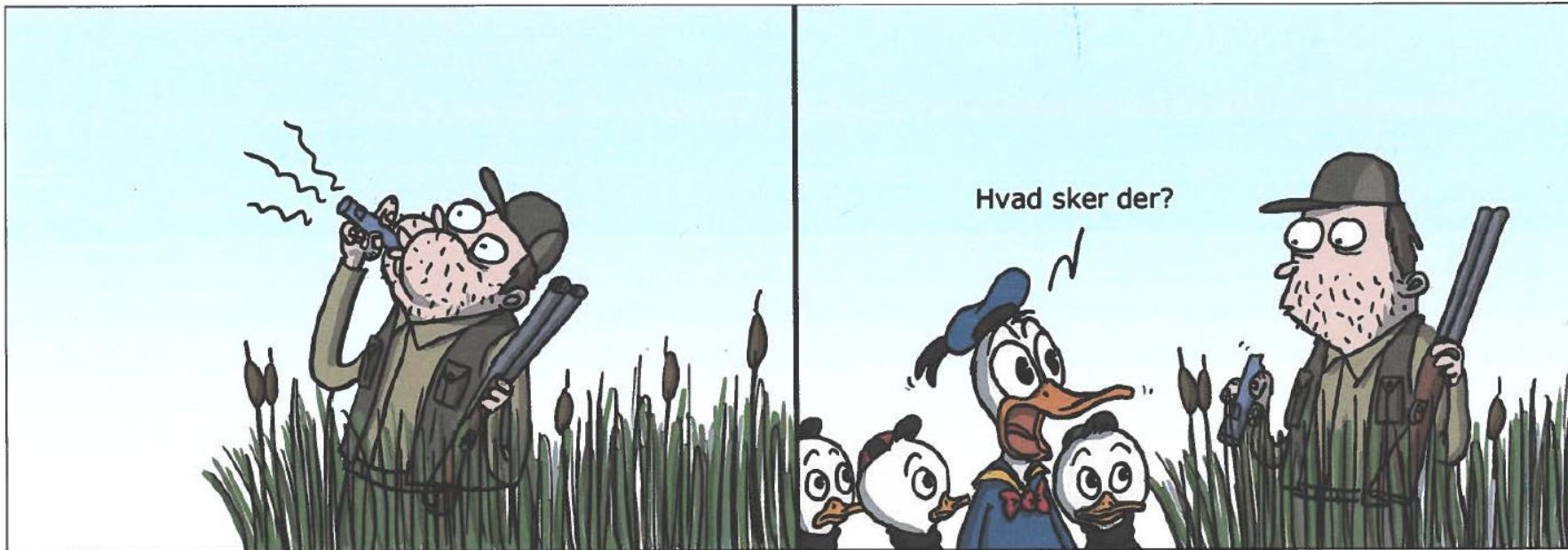
- PUSH de anvendte registre i starten af ISR
POP dem tilbage igen inden RETI

1 .INCLUDE "M32DEF.INC"	18 OUT TIMSK,R20
2 .ORG 0x0 ;location for reset	19 SEI
3 JMP MAIN	20 LDI R20,0xFF
4 .ORG 0x14 ;Timer0 compare match	21 OUT DDRC,R20
5 JMP T0_CM_ISR	22 OUT DDRD,R20
6 ;-----main program-----	23 LDI R20, 0
7 .ORG 0x100	24 HERE: OUT PORTC,R20
8 MAIN: LDI R20,HIGH(RAMEND)	25 INC R20
9 OUT SPH,R20	26 JMP HERE
10 LDI R20,LOW(RAMEND)	27 ;-----ISR for Timer0
11 OUT SPL,R20 ;set up stack	28 T0_CM_ISR:
12 SBI DDRB,5 ;PB5 = output	29 PUSH R20 ;save R20
13 LDI R20,160	30 IN R20,PIND
14 OUT OCR0,R20	31 INC R20
15 LDI R20,0x09	32 OUT PORTD,R20
16 OUT TCCR0,R20	33 POP R20 ;restore R20
17 LDI R20,(1<<OCIE0)	34 RETI

LAB-øvelse 10



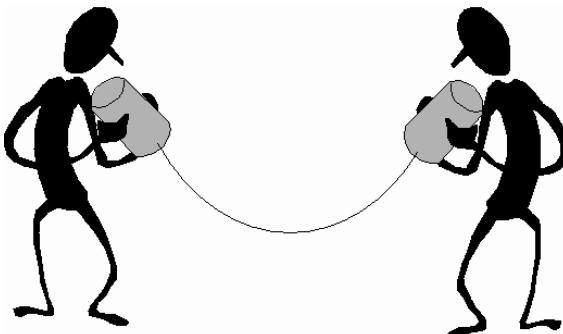
Slut på lektion 16



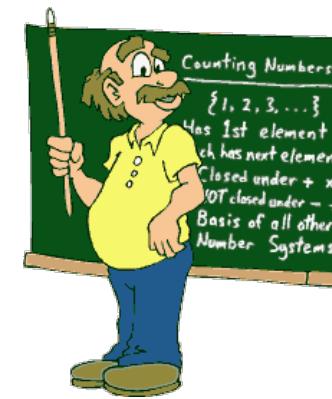
MSYS

Microcontroller Systems

Lektion 17: Seriel kommunikation

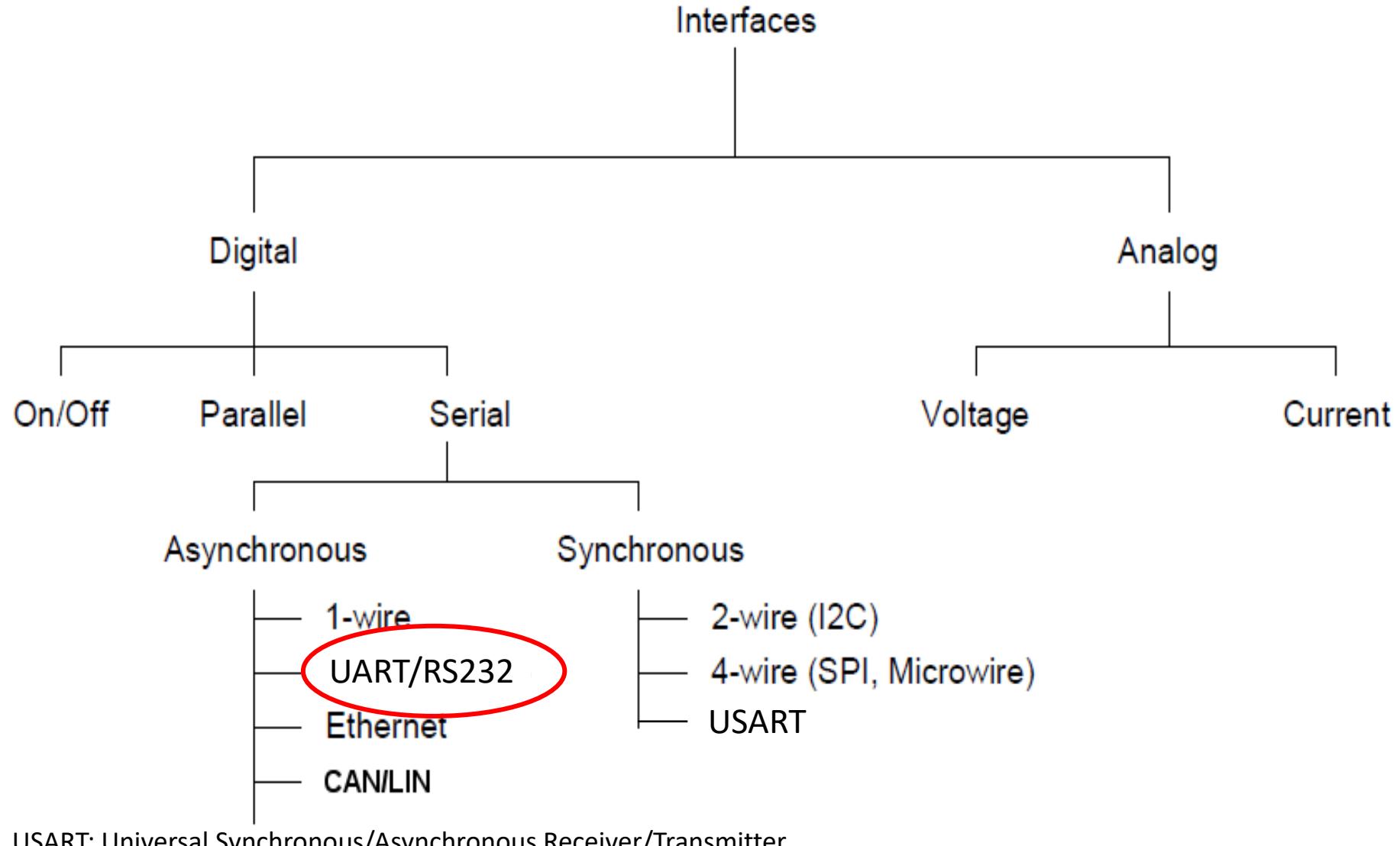


Slides from Henning Hargaard

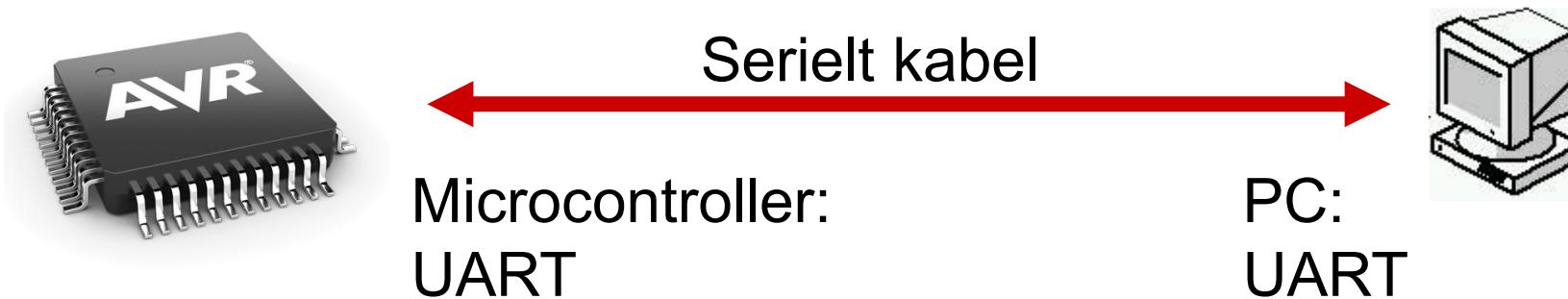


Michael Alrøe

Interfaces

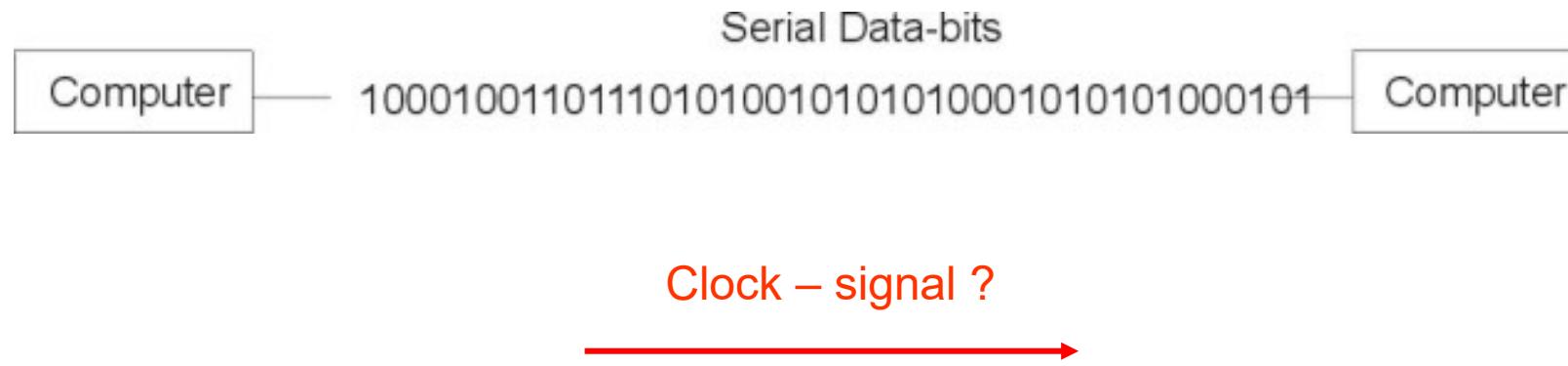


Seriell communication



- Til udveksling af data mellem 2 systemer (f.eks. mellem en PC og en microcontroller) anvendes ofte **seriel/UART kommunikation**.
- Hvert system må have en **UART**.
- Mega2560 har 4 indbyggede UART'er.

Seriell kommunikation

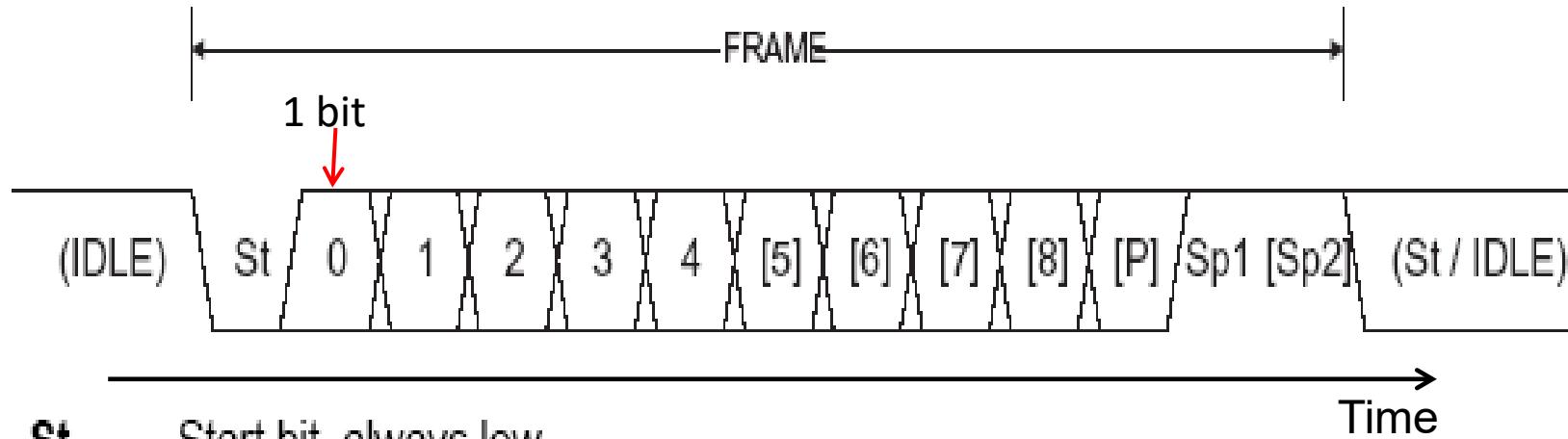


Der er to problemer :

- * Bit -synkronisering.
- * Byte ("frame") –synkronisering.

Asynkront format

- Asynchronous communication:



St Start bit, always low.

(n) Data bits (0 to 8). ← LSB sendes først

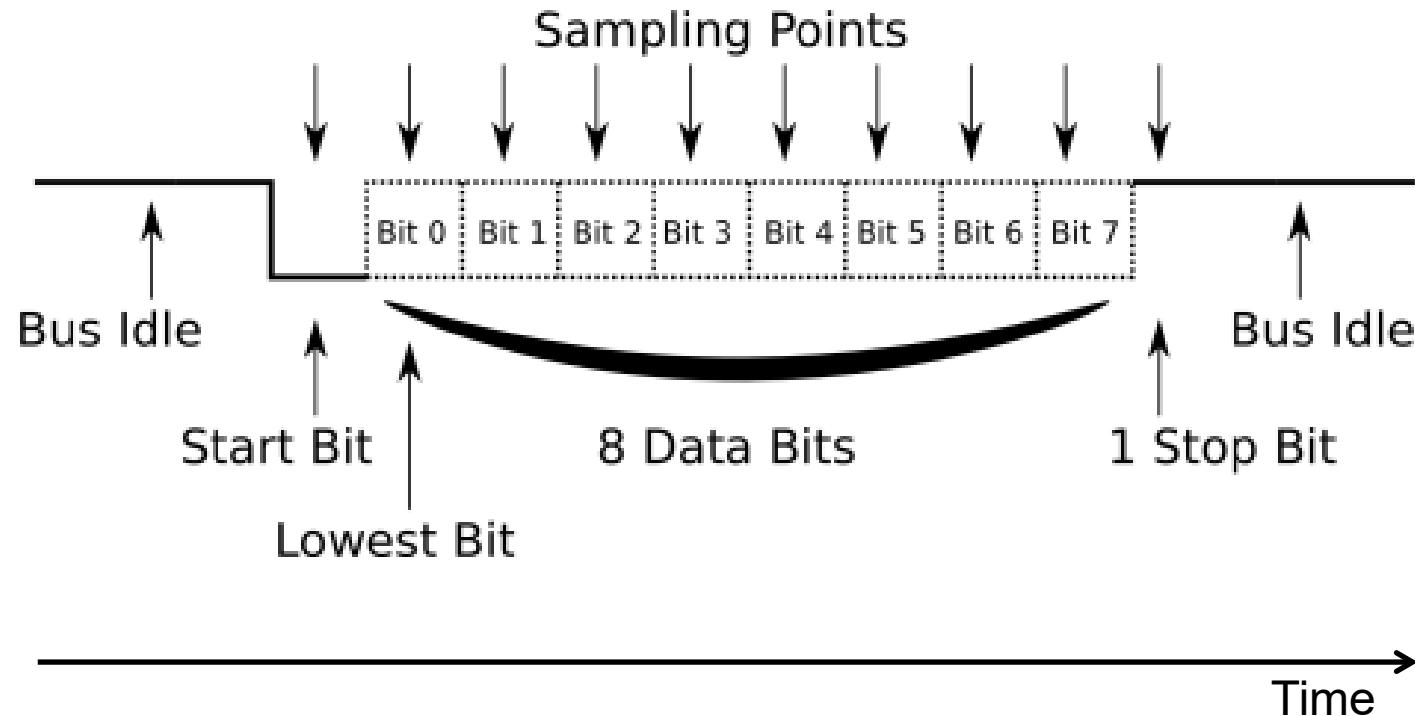
P Parity bit. Can be odd or even. ← Kan udelades -> "None"

Sp Stop bit, always high.

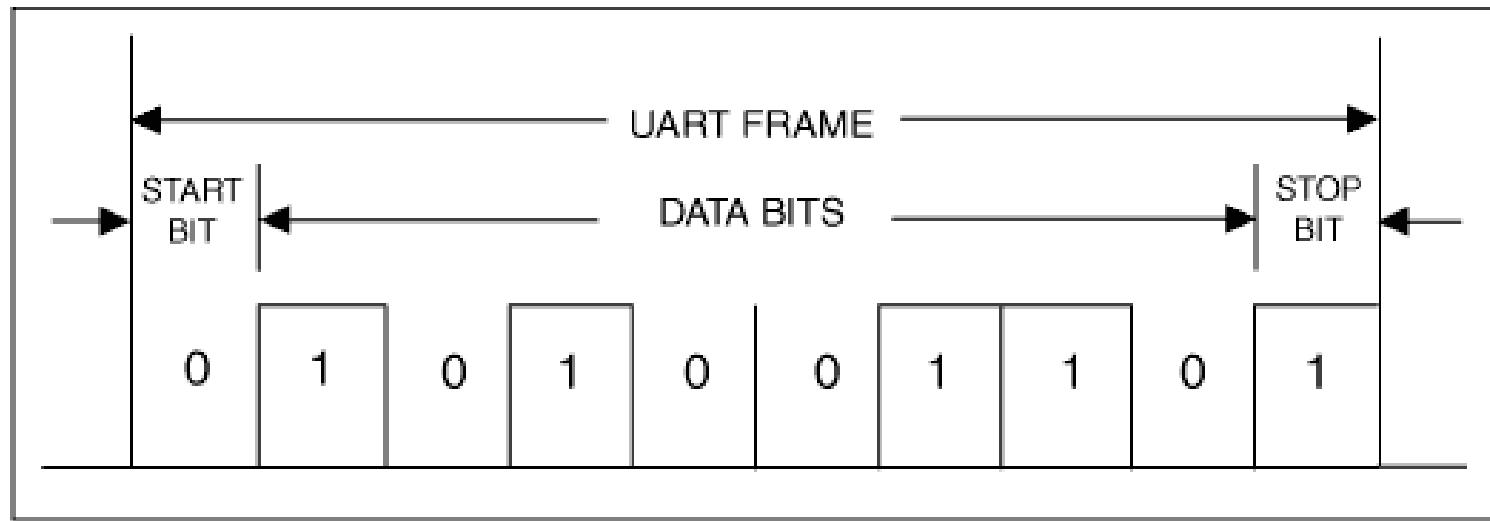
Note: [] indicates optionally

Asynkront format (uden paritet)

UART with 8 Databits, 1 Stopbit and **no Parity**



Eksempel



Hvilken talværdi
sendes ?

ASCII TABLE - "Blot" en kode/aftale for bogstaver

Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	[NULL]	48	30	110000	60	0	96	60	11000000	140	`
1	1	1	1	[START OF HEADING]	49	31	110001	61	1	97	61	11000001	141	a
2	2	10	2	[START OF TEXT]	50	32	110010	62	2	98	62	1100010	142	b
3	3	11	3	[END OF TEXT]	51	33	110011	63	3	99	63	1100011	143	c
4	4	100	4	[END OF TRANSMISSION]	52	34	110100	64	4	100	64	1100100	144	d
5	5	101	5	[ENQUIRY]	53	35	110101	65	5	101	65	1100101	145	e
6	6	110	6	[ACKNOWLEDGE]	54	36	110110	66	6	102	66	1100110	146	f
7	7	111	7	[BELL]	55	37	110111	67	7	103	67	1100111	147	g
8	8	1000	10	[BACKSPACE]	56	38	111000	70	8	104	68	1101000	150	h
9	9	1001	11	[HORIZONTAL TAB]	57	39	111001	71	9	105	69	1101001	151	i
10	A	1010	12	[LINE FEED]	58	3A	111010	72	:	106	6A	1101010	152	j
11	B	1011	13	[VERTICAL TAB]	59	3B	111011	73	;	107	6B	1101011	153	k
12	C	1100	14	[FORM FEED]	60	3C	111100	74	<	108	6C	1101100	154	l
13	D	1101	15	[CARRIAGE RETURN]	61	3D	111101	75	=	109	6D	1101101	155	m
14	E	1110	16	[SHIFT OUT]	62	3E	111110	76	>	110	6E	1101110	156	n
15	F	1111	17	[SHIFT IN]	63	3F	111111	77	?	111	6F	1101111	157	o
16	10	10000	20	[DATA LINK ESCAPE]	64	40	1000000	100	@	112	70	1110000	160	p
17	11	10001	21	[DEVICE CONTROL 1]	65	41	1000001	101	A	113	71	1110001	161	q
18	12	10010	22	[DEVICE CONTROL 2]	66	42	1000010	102	B	114	72	1110010	162	r
19	13	10011	23	[DEVICE CONTROL 3]	67	43	1000011	103	C	115	73	1110011	163	s
20	14	10100	24	[DEVICE CONTROL 4]	68	44	1000100	104	D	116	74	1110100	164	t
21	15	10101	25	[NEGATIVE ACKNOWLEDGE]	69	45	1000101	105	E	117	75	1110101	165	u
22	16	10110	26	[SYNCHRONOUS IDLE]	70	46	1000110	106	F	118	76	1110110	166	v
23	17	10111	27	[ENG OF TRANS. BLOCK]	71	47	1000111	107	G	119	77	1110111	167	w
24	18	11000	30	[CANCEL]	72	48	1001000	110	H	120	78	1111000	170	x
25	19	11001	31	[END OF MEDIUM]	73	49	1001001	111	I	121	79	1111001	171	y
26	1A	11010	32	[SUBSTITUTE]	74	4A	1001010	112	J	122	7A	1111010	172	z
27	1B	11011	33	[ESCAPE]	75	4B	1001011	113	K	123	7B	1111011	173	{
28	1C	11100	34	[FILE SEPARATOR]	76	4C	1001100	114	L	124	7C	1111100	174	
29	1D	11101	35	[GROUP SEPARATOR]	77	4D	1001101	115	M	125	7D	1111101	175	}
30	1E	11110	36	[RECORD SEPARATOR]	78	4E	1001110	116	N	126	7E	1111110	176	-
31	1F	11111	37	[UNIT SEPARATOR]	79	4F	1001111	117	O	127	7F	1111111	177	[DEL]
32	20	100000	40	[SPACE]	80	50	1010000	120	P					
33	21	100001	41	!	81	51	1010001	121	Q					
34	22	100010	42	"	82	52	1010010	122	R					
35	23	100011	43	#	83	53	1010011	123	S					
36	24	100100	44	\$	84	54	1010100	124	T					
37	25	100101	45	%	85	55	1010101	125	U					
38	26	100110	46	&	86	56	1010110	126	V					
39	27	100111	47	'	87	57	1010111	127	W					
40	28	101000	50	(88	58	1011000	130	X					
41	29	101001	51)	89	59	1011001	131	Y					
42	2A	101010	52	*	90	5A	1011010	132	Z					
43	2B	101011	53	+	91	5B	1011011	133	[
44	2C	101100	54	,	92	5C	1011100	134	\					
45	2D	101101	55	-	93	5D	1011110	135]					
46	2E	101110	56	.	94	5E	1011110	136	^					
47	2F	101111	57	/	95	5F	1011111	137	_					

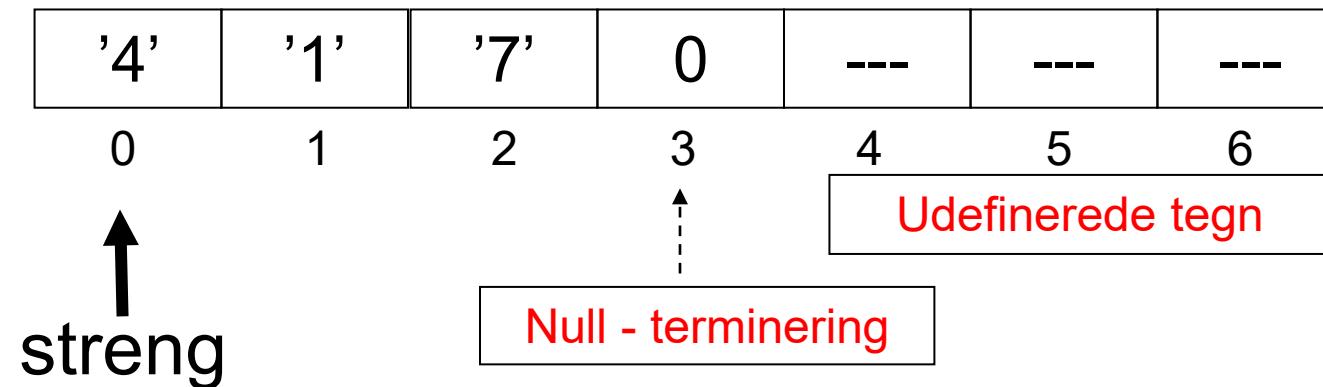
itoa(int n, char* str)

Hvis man inkluderer `<stdlib.h>`, har man adgang til en række funktioner, hvoraf `itoa()` er en af dem.

`itoa(int val, char *s, int radix)` vil konvertere `val`'s talværdi til en 0-termineret tekst i et array, som pointeren `s` peger på.
`radix` er normalt 10 -> 10-tals system!

Eksempel:

```
uint16_t tal = 417;  
char streng[7];  
itoa( tal, streng, 10 );
```



Baud Rate

- Baud Rate = Antal bits per sekund.
- Baud Rate = 1 / bit-tiden.
- Eksempel : 9600 Baud ~ bit-tid = 104 uS.
- Standard Baud Rates :
110, 300, 600, 1200, 2400, 4800
9600, 14400, 19200
38400, 57600, **115200**

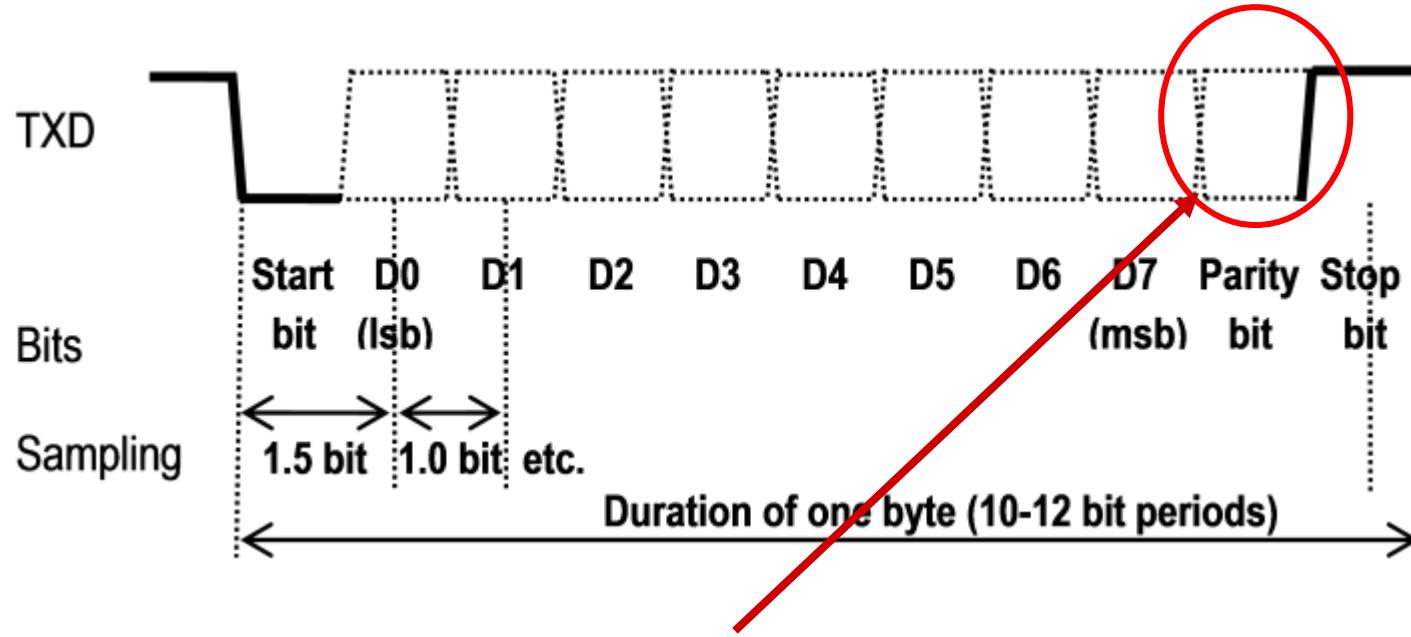
Test ("socrative.com"): Room = AUQUIZ

Hvor lang tid tager det at sende et tegn via en UART,
når der anvendes:

- * Baud rate = 9600 bit/s
 - * 1 startbit
 - * 8 databit
 - * Ingen paritet
 - * 1 stopbit
-
- A: Cirka 10 ms.
 - B: Cirka 100 us.
 - C: Cirka 1 ms.
 - D: Cirka 10 us.

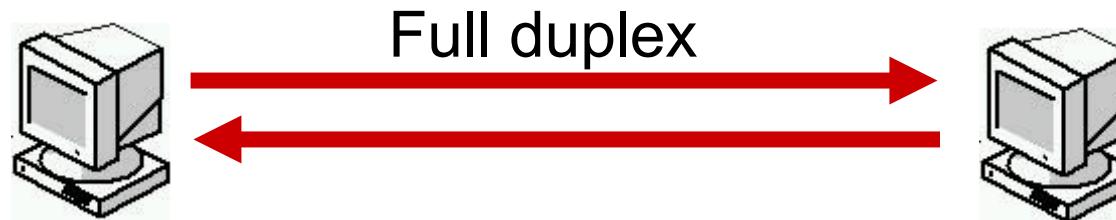
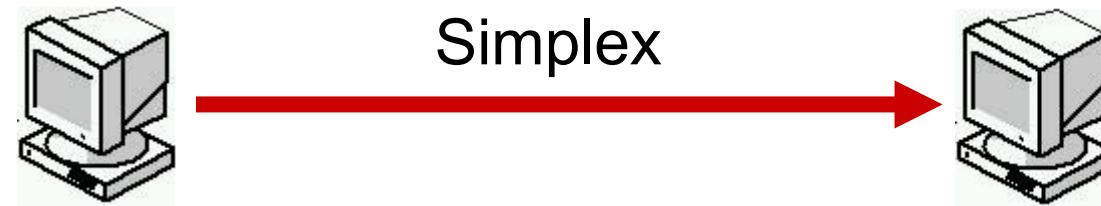


Paritet

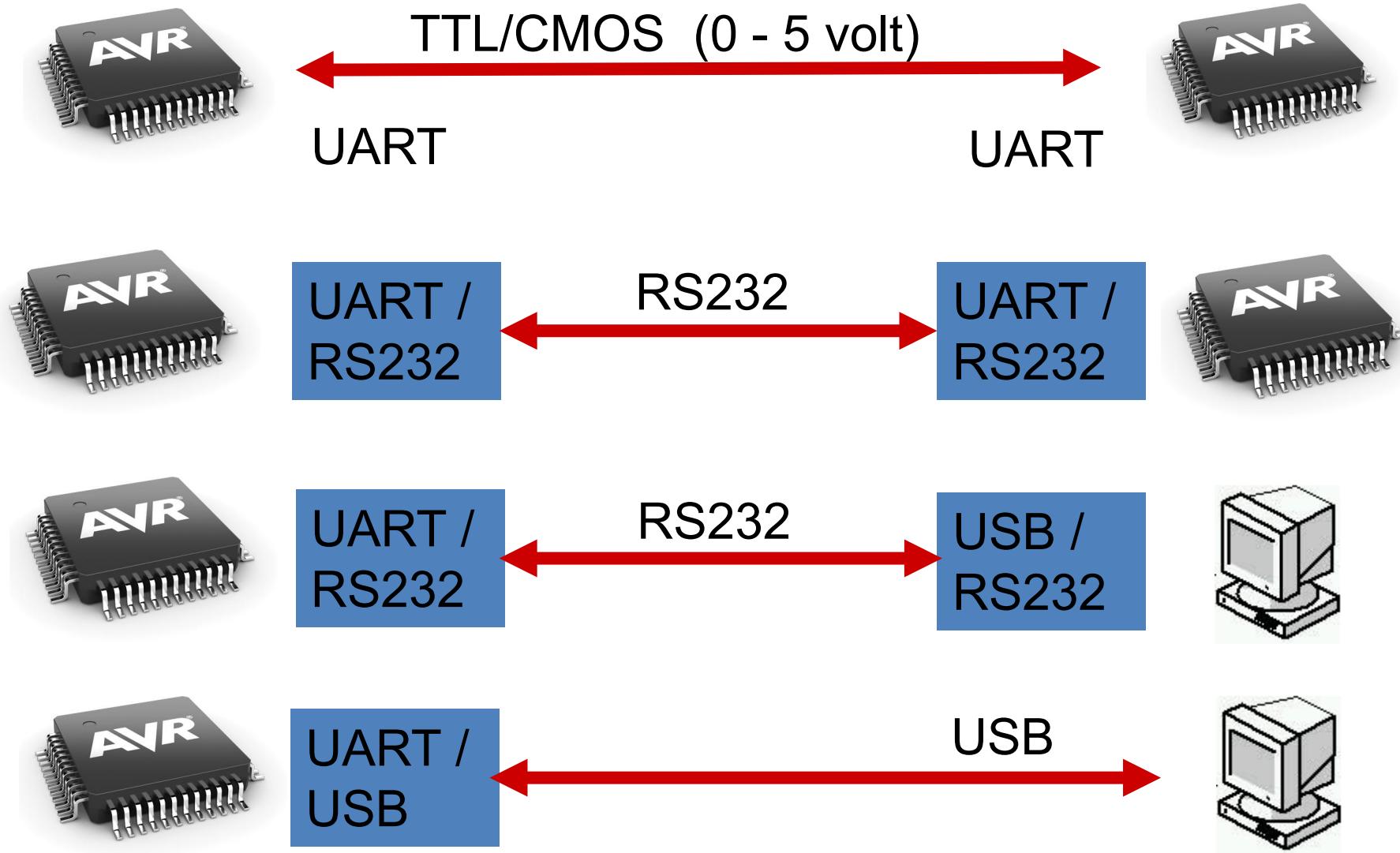


- “**Even parity**” : Antallet af 1-taller (i data og P-bit) skal være et **lige** tal.
- “**Odd parity**” : Antallet af 1-taller (i data og P-bit) skal være **ulige** tal.
- “**No parity**” anvendes oftest (ingen paritets-bit).

Simplex / duplex



(Kommunikations-standarder)



(Ofte anvendt standard: RS232)

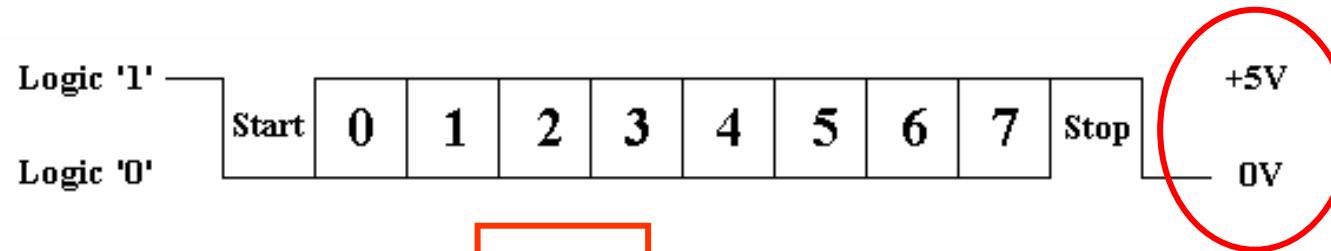


Figure 4 TTL/CMOS Serial Logic Waveform

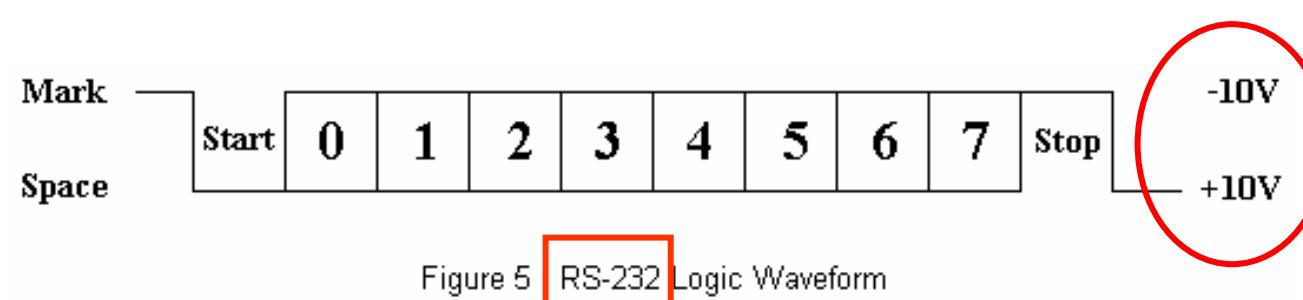
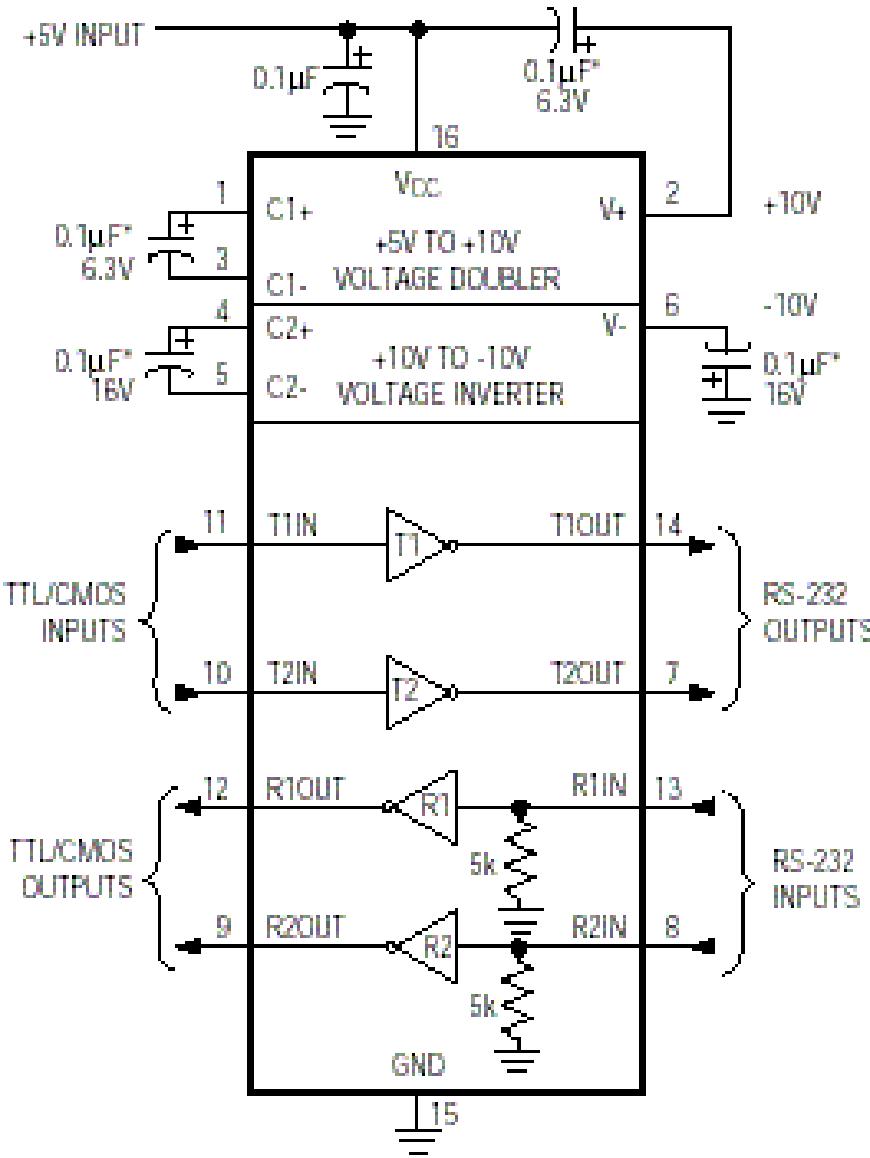


Figure 5 RS-232 Logic Waveform

- Logisk 0 : +3 til +25 volt.
- Logisk 1 : -3 til -25 volt.

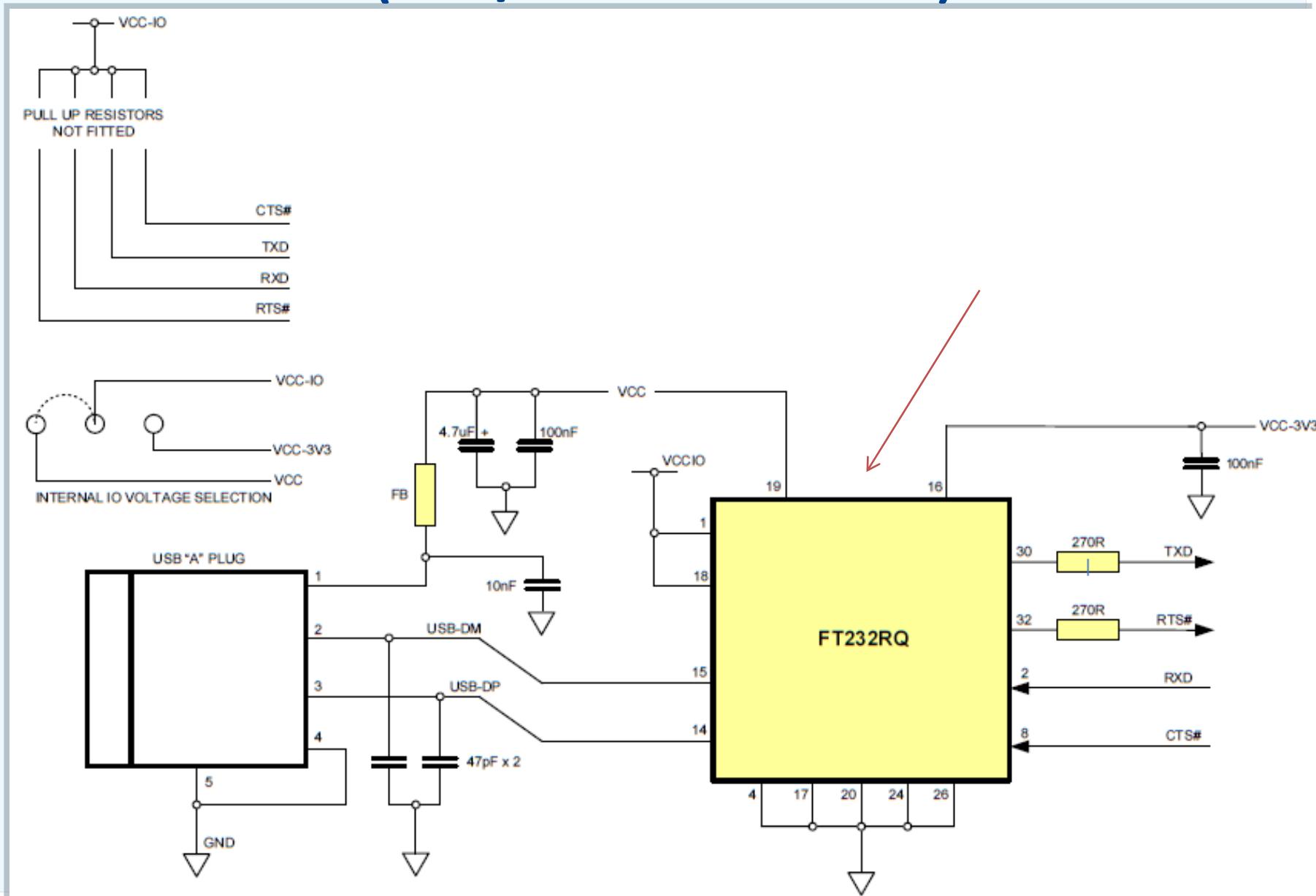
(MAX232 tranceiver-kreds)



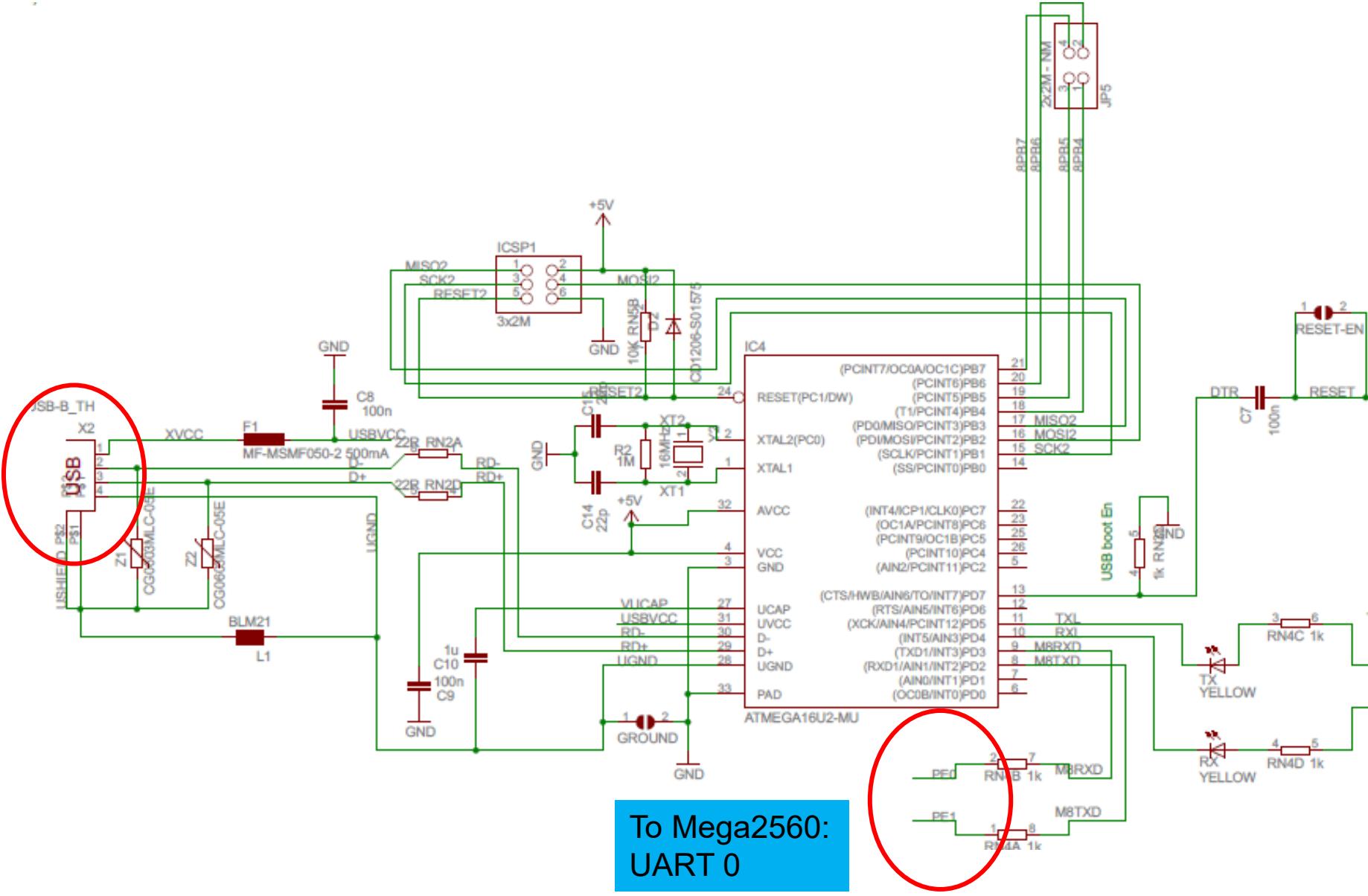
Pris ~ 1-2 kroner



(USB/UART Converter)



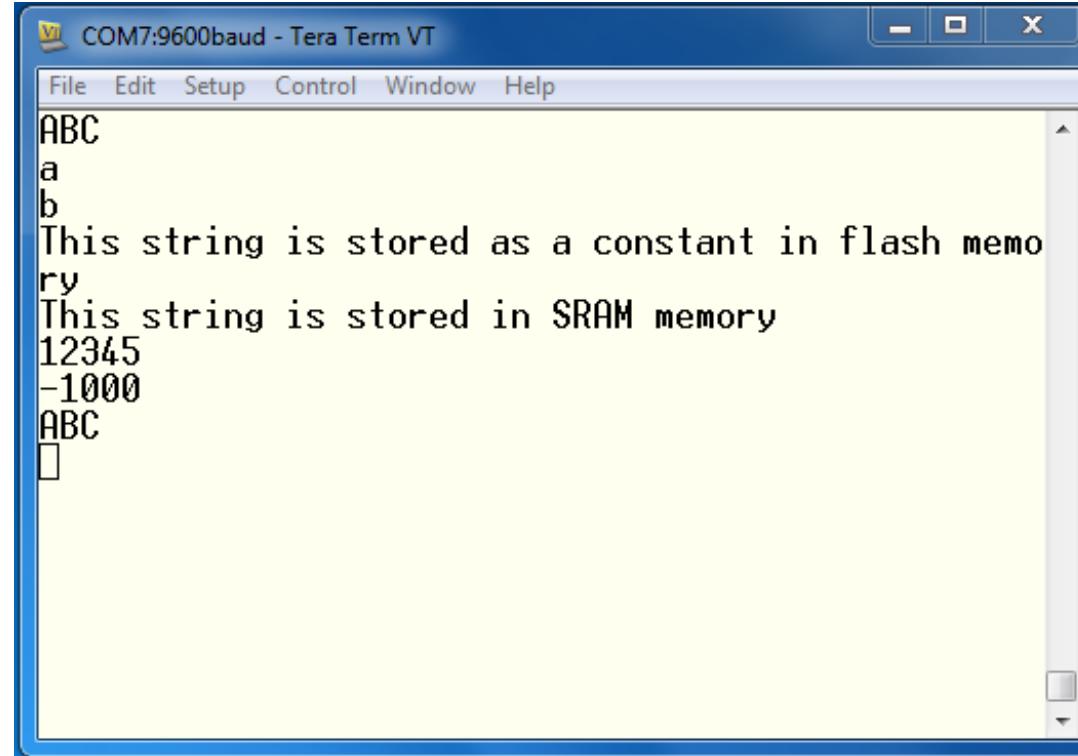
(USB/UART konverter i Arduino Mega2560)



USB
connector

To Mega2560:
UART 0

PC: "Tera Terminal"

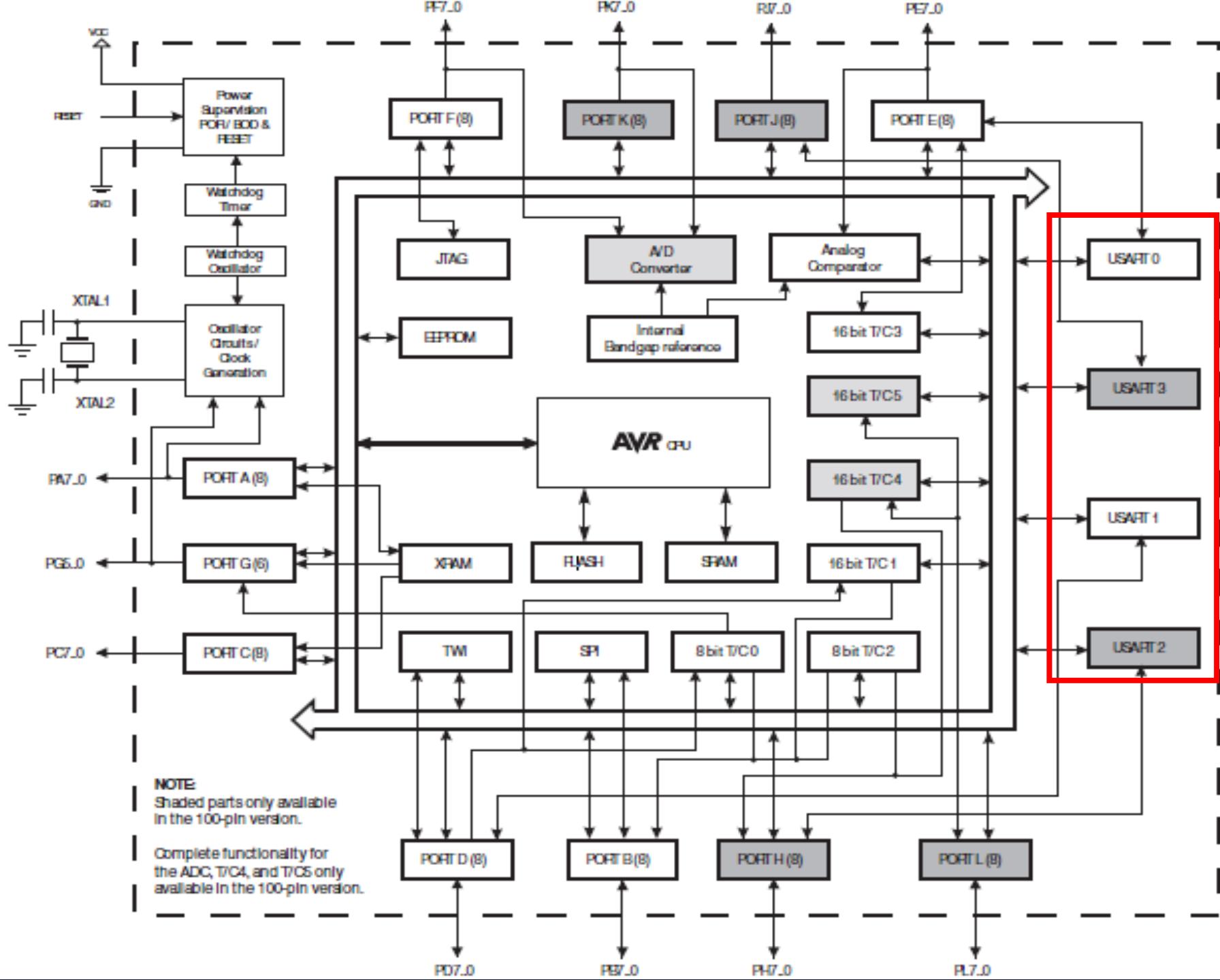


På PC'en startes et **terminalprogram** (f.eks. det gratis program "Tera Terminal").

Når vi **trykker en tast på PC tastaturet**, sendes tegnets ASCII-kode ud på COM-porten.

Hvis PC'ens COM-port **modtager et tegn**, vises det i vinduet.

Mega2560 - 4 USART'er



Mega2560: USART pins

USART0 :

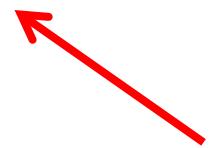
TXD0 = PE, ben 1

RXD0 = PE, ben 0

USART1 :

TXD1 = PD, ben 3

RXD1 = PD, ben 2



Via UART bridge til Arduino Mega2560
USB - stikket

USART2 :

TXD2 = PH, ben 1

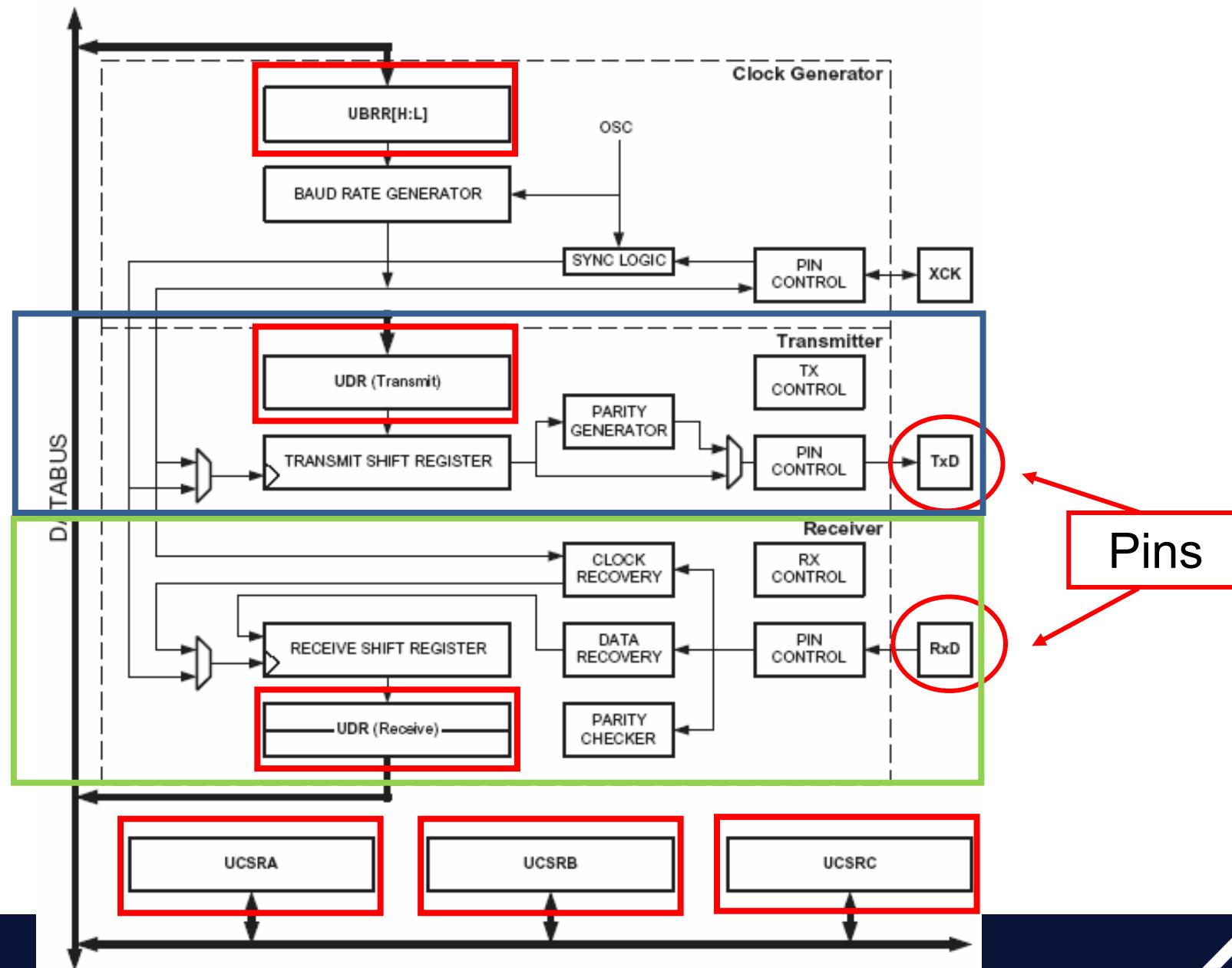
RXD2 = PH, ben 0

USART3 :

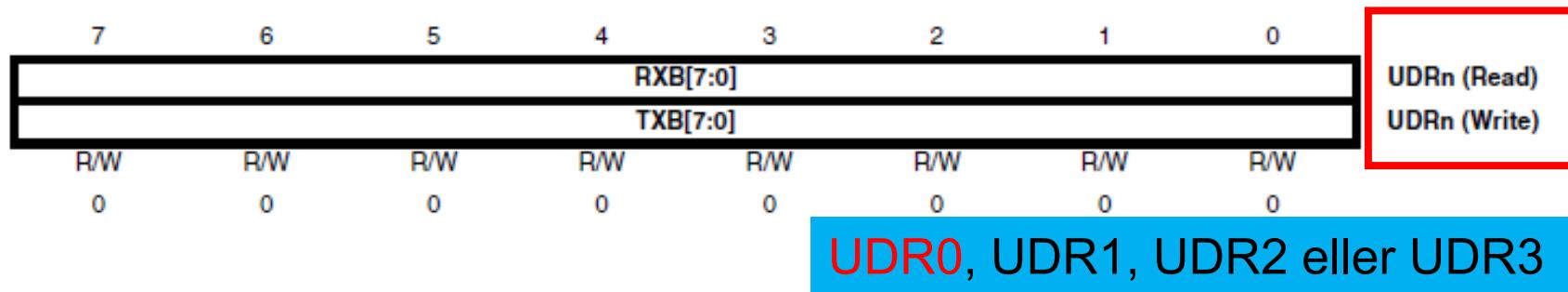
TXD3 = PJ, ben 1

RXD3 = PJ, ben 0

Mega32/Mega2560 USART



Mega2560: UDRn. Usart Data Registers



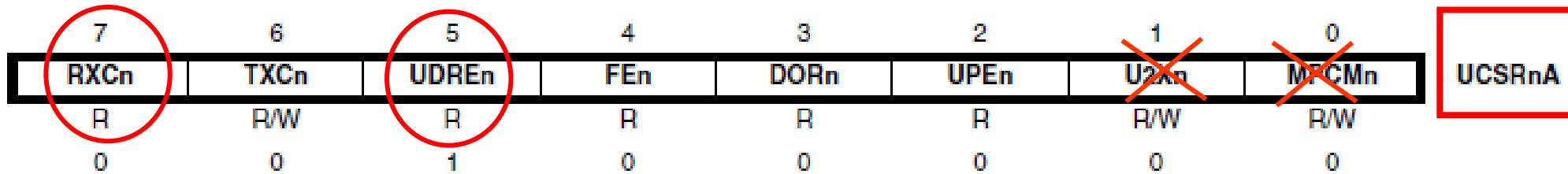
Bemærk: Fysisk to forskellige registre med samme navn (UDRn)

char x;

Læsning af modtaget tegn : **x = UDR0;**

Send tegn : **UDR0 = x;**

Mega2560: UCSRnA: Control and Status Registers A



UCSR0A, UCSR1A, UCSR2A eller UCSR3A

RXCn -> 1: Nyt tegn modtaget ("kan hentes i UDR").

TXC -> 1: "Sender tom" (klar til at sende nyt tegn og sende skifteredisteret tomt).

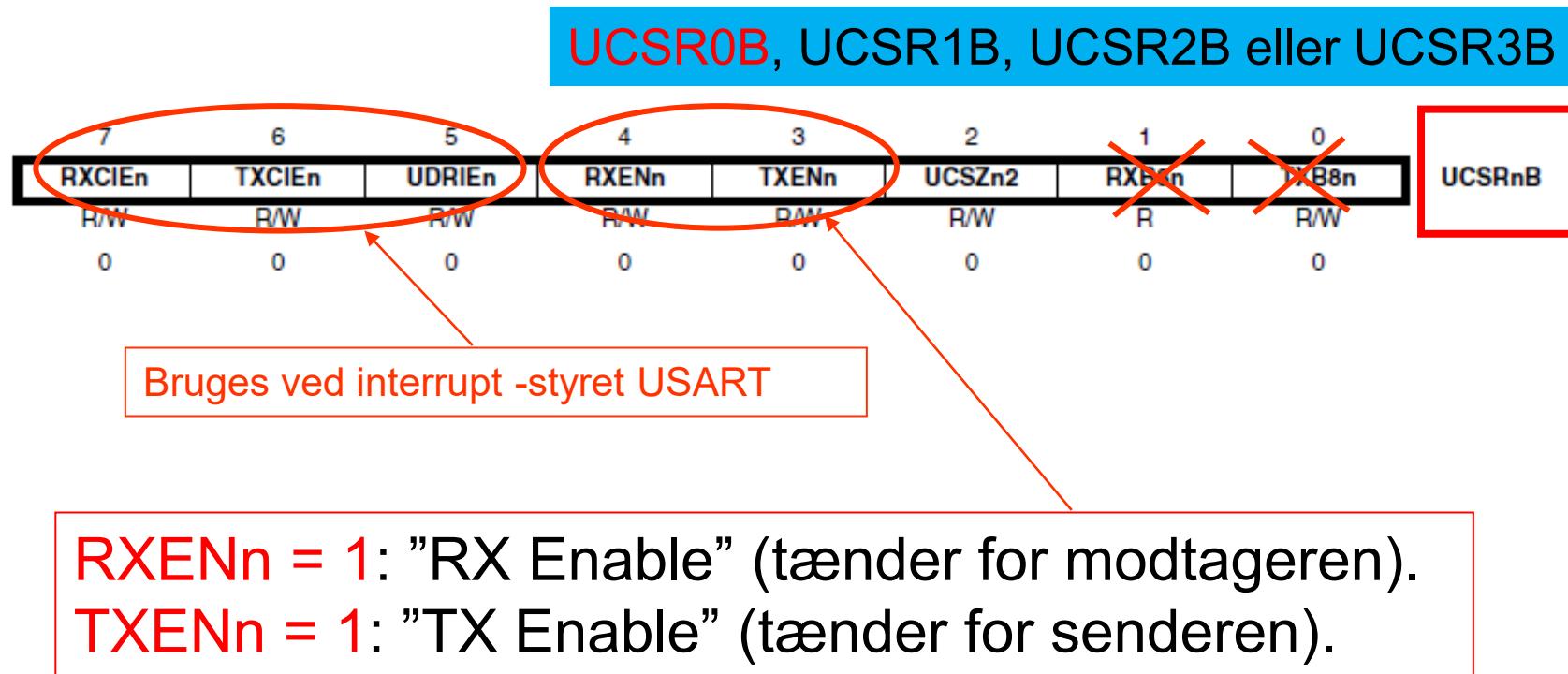
UDREn ->1: Klar til at sende nyt tegn ("der må skrives til UDR").

FEn -> 1: "Framing Error" (modtaget tegn har fejl i stop bit).

DORn -> 1: "Data overrun" (tegn modtaget, inden foregående er blevet læst af SW).

PEn -> 1: "Parity Error" (modtaget tegn har paritetsfejl).

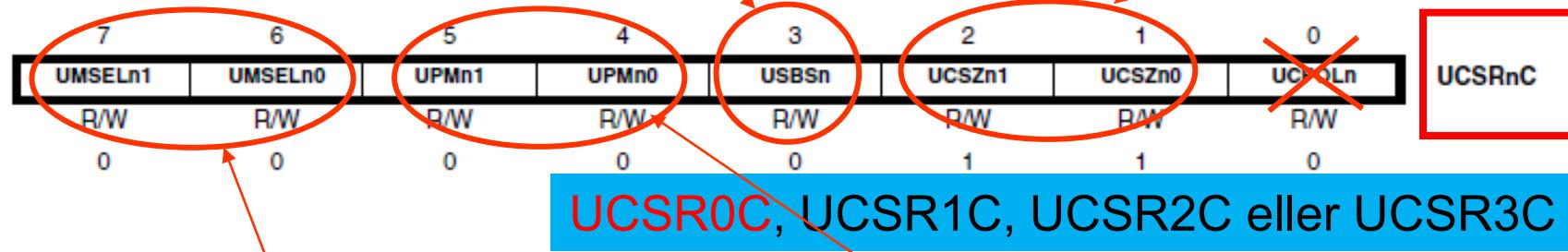
Mega2560: UCSRnB. Control and Status Register B



Mega2560: UCSRnC. Control and Status Register C

UCSZn1 og UCSZn0 : Antal data bits (se næste side).

USBSn : 0 = 1 stop bit. 1 = 2 stop bits.



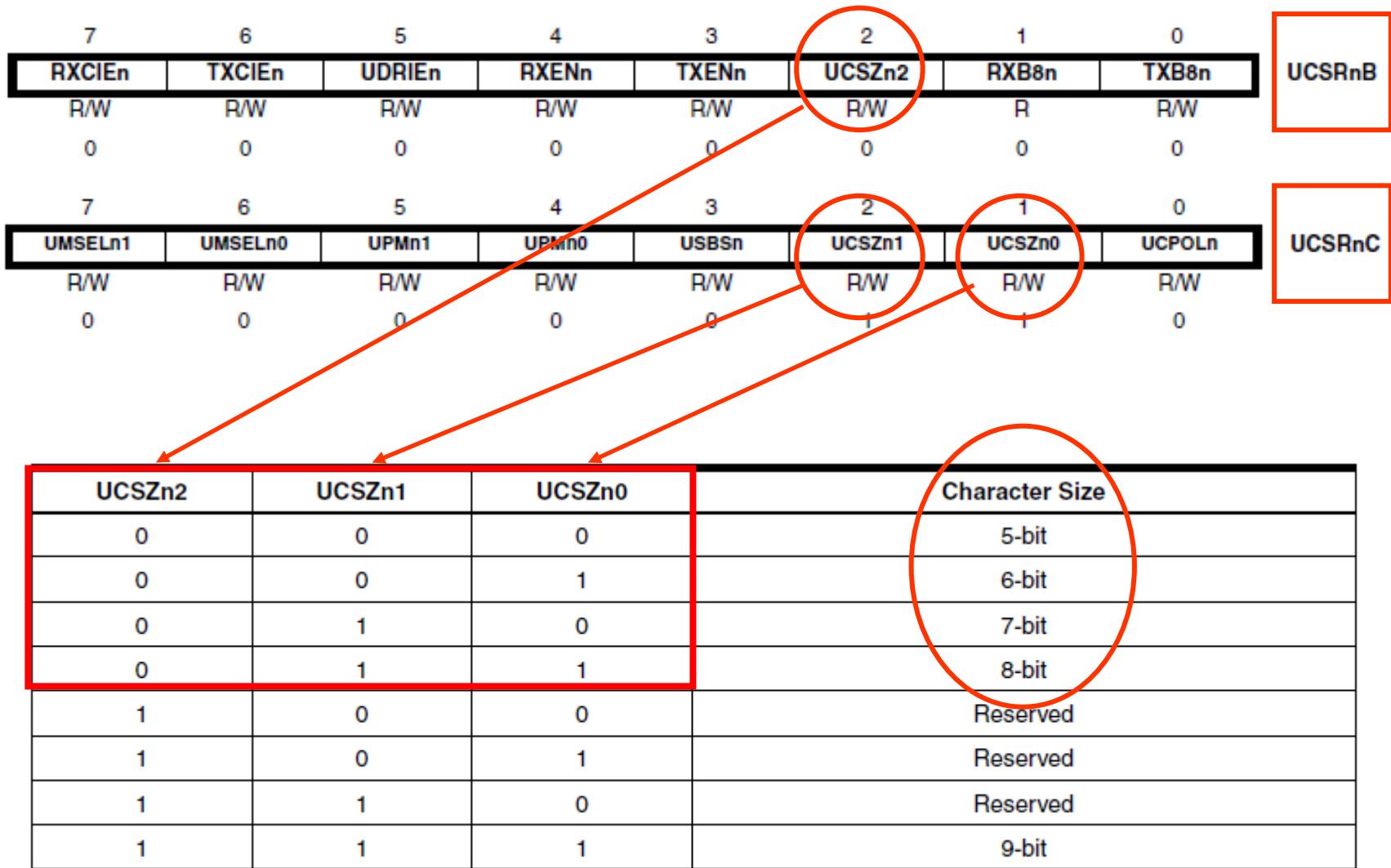
UCSR0C, UCSR1C, UCSR2C eller UCSR3C

UMSELn1 = 0 og UMSELn0 = 0 :
Asynkron mode !

UPMn1 og UPMn0: Valg af paritet:

UPMn1	UPMn0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

Mega2560: Antal data bits



Test ("socrative.com"): Room = AUQUIZ

Hvilket bit i registeret UCSRnA vil gå højt, når
UART'en har modtaget et nyt tegn ?

- A: Bit 7: RXC
- B: Bit 6: TXC
- C: Bit 5: UDRE
- D: Bit 4: FE
- E: Bit 2: PE



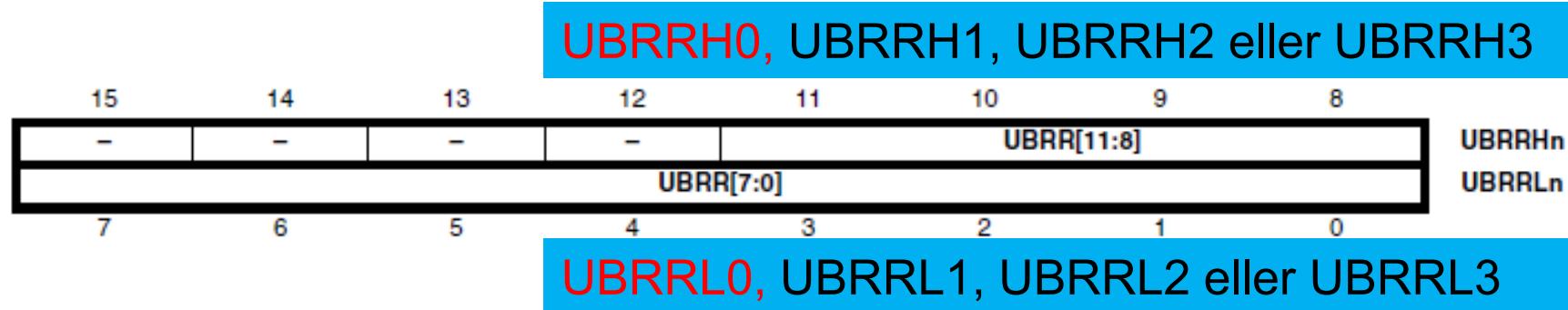
Test ("socrative.com"): Room = AUQUIZ

Før man må sende et nyt tegn via UART 0, skal man sikre sig, at senderen er klar til at modtage tegnet.
Hvilken metode er korrekt ?

- A: `while ((UCSR0A | 0b00100000) == 0)
{ }`
- B: `while ((UCSR0A & 0b00100000) == 0)
{ }`
- C: `while ((UCSR0A & 0b00100000) != 0)
{ }`
- D: `while (UDR0 != 0)
{ }`



Mega2560: UBRRHn + UBRLLn. Baud Rate Registre



Operating Mode	Equation for Calculating Baud Rate ⁽¹⁾	Equation for Calculating UBRR Value
Asynchronous Normal mode (U2Xn = 0)	$BAUD = \frac{f_{OSC}}{16(UBRRn + 1)}$	$UBRRn = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed mode (U2Xn = 1)	$BAUD = \frac{f_{OSC}}{8(UBRRn + 1)}$	$UBRRn = \frac{f_{OSC}}{8BAUD} - 1$

"UBRRn" = (256 * UBRRHn) + UBRLLn

(Baud rate afrundingsfejl ved 16 MHz)

Baud Rate [bps]	$f_{osc} = 16.0000\text{MHz}$			
	U2Xn = 0		U2Xn = 1	
	UBRR	Error	UBRR	Error
2400	416	-0.1%	832	0.0%
4800	207	0.2%	416	-0.1%
9600	103	0.2%	207	0.2%
14.4K	68	0.6%	138	-0.1%
19.2K	51	0.2%	103	0.2%
28.8K	34	-0.8%	68	0.6%
38.4K	25	0.2%	51	0.2%
57.6K	16	2.1%	34	-0.8%
76.8K	12	0.2%	25	0.2%
115.2K	8	-3.5%	16	2.1%
230.4K	3	8.5%	8	-3.5%
250K	3	0.0%	7	0.0%
0.5M	1	0.0%	3	0.0%
1M	0	0.0%	1	0.0%
Max. ⁽¹⁾	1Mbps		2Mbps	

$$\text{"UBRR"} = (256 * \text{UBRRH}) + \text{UBRRL}$$

Test ("socrative.com"): Room = AUQUIZ

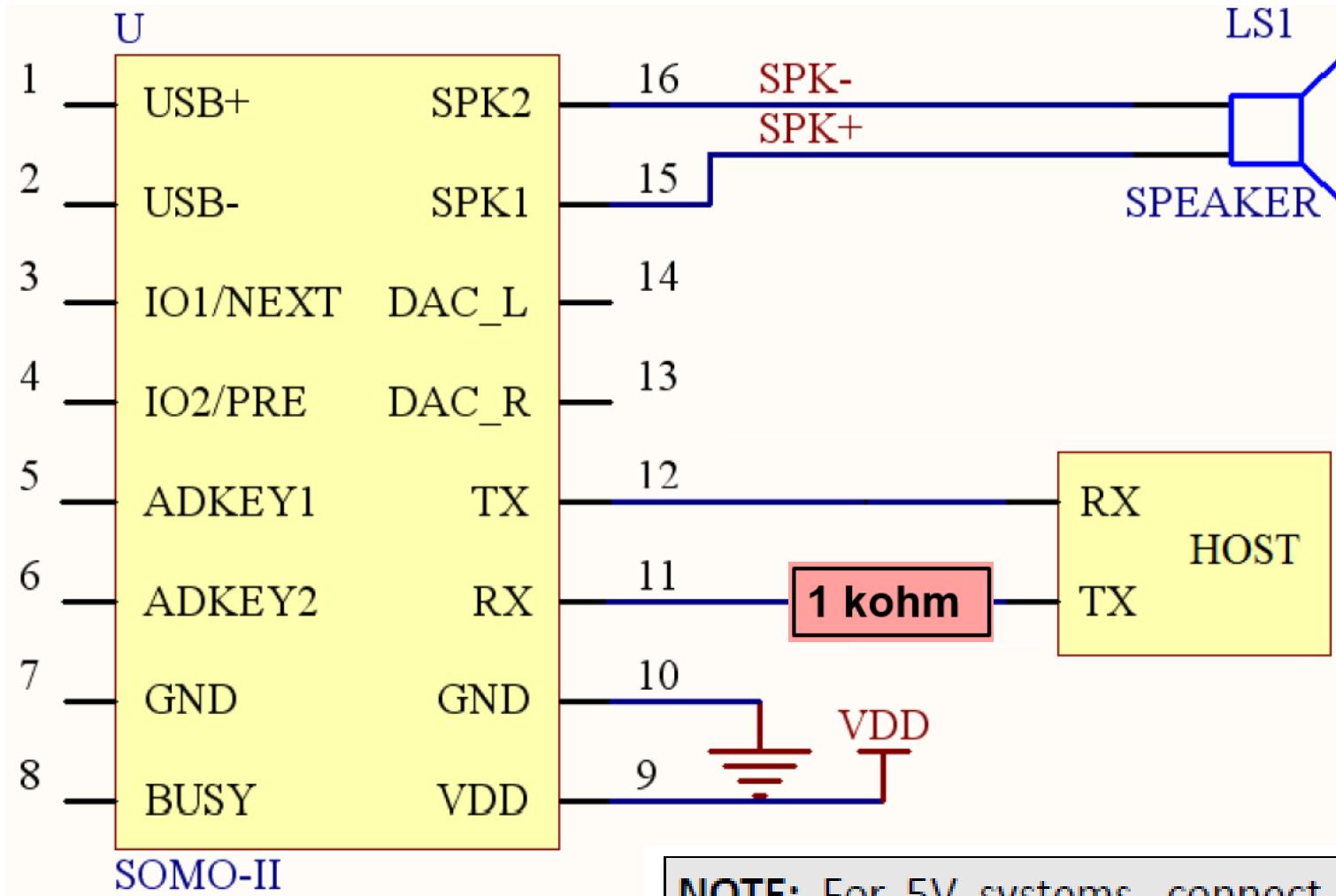
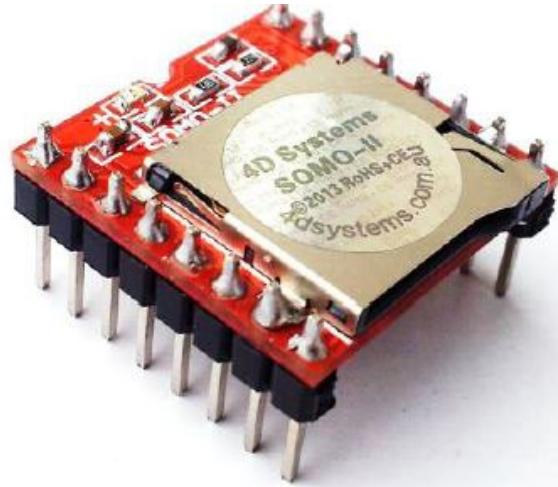
Under initiering af en Mega2560's UART 1 skrives følgende værdi til UBRR1:
UBRR1 = 3332;

Mega2560's CPU clockfrekvens er 16 MHz.
Hvilken baud rate anvendes ?

- A: 300 bit/s
- B: 9600 bit/s
- C: 115200 bit/s
- D: 1200 bit/s



Eksempel: SOMO-II MP3 player



NOTE: For 5V systems, connect a series resistor (1K Ohms) on the RX line of the SOMO-II to the TX of the 5V micro-controller.

Eksempel: SOMO-II MP3 player

FORMAT: \$S, CMD, Feedback, Para1, Para2, Checksum1, Checksum2, \$0		
\$S	Start Character \$S is 0x7E in HEX	Every command starts with this
CMD	Command Code	Every command has a unique command code, which determines the operation
Feedback	Command Feedback	Specifies whether feedback is required by the host microcontroller in reply to the command. 1 = Feedback, 0 = No Feedback
Para1	Parameter #1	First parameter of the specific Command Code
Para2	Parameter #2	Second parameter of the specific Command Code
Checksum1	Checksum #1	First byte of the checksum. Checksum calculation shown below.
Checksum2	Checksum #2	Second byte of the checksum. Checksum calculation shown below.
\$0	End Character \$0 is 0xEF in HEX	Every command ends with this

Checksum Calculation:

The checksum is calculated using the following formula.

$$\text{Checksum (2 bytes)} = \text{0xFFFF} - (\text{CMD} + \text{Feedback} + \text{Para1} + \text{Para2}) + 1$$



Eksempel: SOMO-II MP3 player

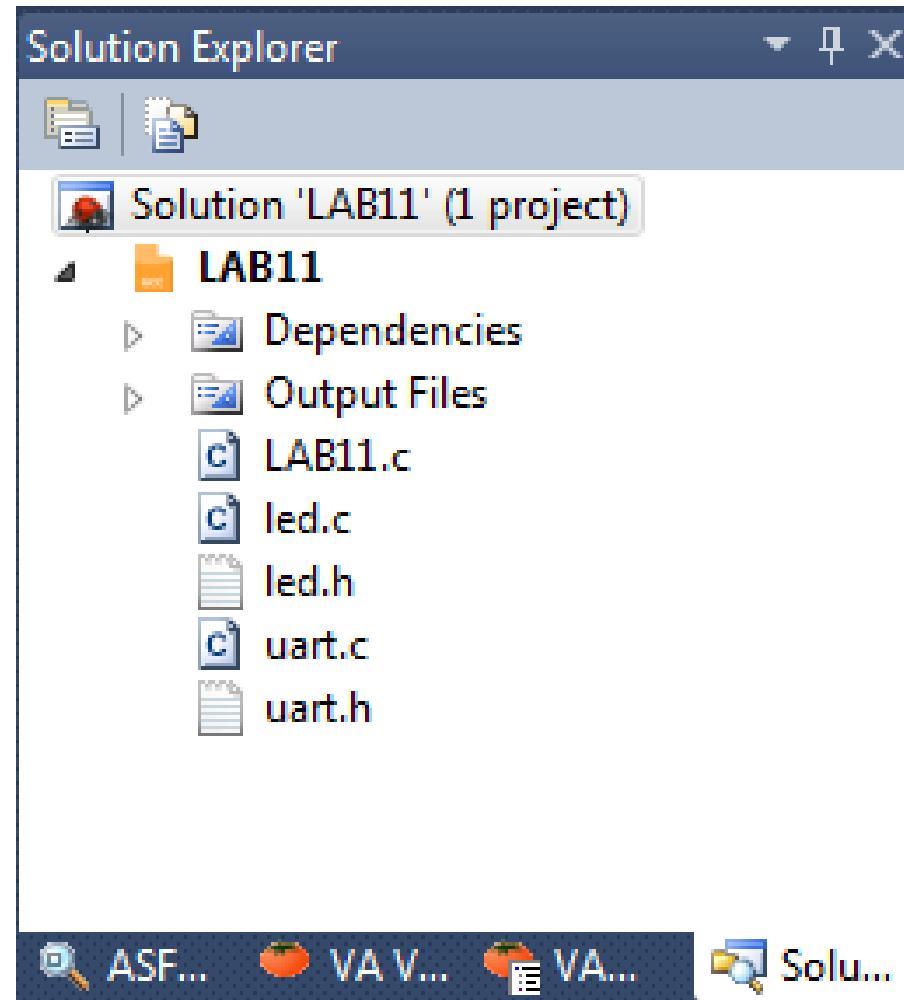
Function	Serial Command	Description
NEXT	7E 01 00 00 00 FF FF EF	If no track is currently playing, issuing the NEXT command will start playing the first track copied to the media (see Section 6). If the SOMO-II is currently playing a song or has previously played a song, this will play the next song in the order copied on to the media.
PREVIOUS	7E 02 00 00 00 FF FE EF	If no track is currently playing, issuing the PREVIOUS command will start playing the last track copied to the media (see Section 6). If the SOMO-II is currently playing a song or has previously played a song, this will play the previous song in the order copied on to the media.
SPECIFY TRACK #	7E 03 00 00 01 FF FC EF 7E 03 00 00 02 FF FB EF 7E 03 00 00 0A FF F3 EF	Start playing the first track copied to the media. (See Section 6) This will start playing the second track copied to the media. This will start playing the tenth track copied to the media.
VOLUME +	7E 04 00 00 00 FF FC EF	This will increase the volume by 1
VOLUME -	7E 05 00 00 00 FF FB EF	This will decrease the volume by 1
VOLUME #	7E 06 00 00 1E FF DC EF 7E 06 00 00 05 FF F5 EF	This will set the volume to be 30 (30 is the Max) This will set the volume to be 5
SPECIFY EQ	7E 07 00 00 01 FF F8 EF 7E 07 00 00 04 FF F5 EF	This will set the EQ to pop This will set the EQ to classic (0/1/2/3/4/5 Normal, Pop, Rock, Jazz, Classic, Bass)
REPEAT A TRACK	7E 08 00 00 01 FF F7 EF 7E 08 00 00 02 FF F6 EF 7E 08 00 00 1F FF D9 EF	This will repeat the first track copied to the media. (See Section 6) This will repeat the second track copied to the media. This will repeat the thirty first track copied to the media.

LAB11: UART driver

```
*****  
* "uart.h": *  
* Header file for Mega2560 UART driver. *  
* Using UART 0. *  
* Henning Hargaard, 5/4 2019 *  
*****/  
  
void InitUART();  
unsigned char CharReady();  
char ReadChar();  
void SendChar(char Tegn);  
void SendString(char* Streng);  
void SendInteger(int Tal);  
*****/
```

Driverens header fil.

LAB11 filer



InitUART()

void InitUART()

Skal initiere UART 0 til:

- Asynkron mode.
- 9600 bit/s
- Ingen paritet
- Både RX og TX enabled.
- 1 stop bit.
- Alle interrupts disabled.

Vi antager, at Mega2560's CPU-clockfrekvens er 16 MHz.

Karakter funktioner

unsigned char CharReady()

Meddeler, om UART 0 har modtaget et tegn.

Hvis et tegn er modtaget, returneres en værdi forskellig fra 0 (= TRUE).

Hvis der ikke er modtaget et tegn, returneres værdien 0 (= FALSE).

Funktionen skal ikke afvente modtagelse af et tegn, men blot returnere oplysningen om, hvorvidt et tegn er modtaget.

char ReadChar()

Returnerer et modtaget tegn fra UART 0's modtageregister (UDR).

Funktionen skal først afvente, at et tegn modtages (bit RXC0 i registeret UCSRA0).

Derefter skal tegnet i UDR0 returneres.

void SendChar(char Tegn)

Sender et tegn via UART 0. Tegnets overføres som parameter.

Inden tegnet skrives til data registeret (UDR0), skal funktionen afvente "UART data register empty" (bit UDRE0 i registeret UCSRA0).

SendString()

void SendString(char* Streng)

Udskriver en 0-termineret tekststreng ved hjælp af UART 0.

Funktion modtager som parameter en pointer til den streng, som vi ønsker udskrevet.

Pointeren peger altid på det første tegn i strengen, som altså er 0-termineret.

Brugeren har på forhånd (altså inden denne funktion kaldes) oprettet og lagret strengen.

Nedenstående viser i pseudo-kode, hvordan funktionen kan implementeres:

```
while ("Det som pointeren peger på" ikke er 0)
{
    SendChar("Det som pointeren peger på");
    Flyt pointeren en plads frem;
}
```

SendInteger()

void SendInteger(int Tal)

Denne funktion skal udskrive værdien af integer "Tal", der modtages som parameter.

Hvis man f.eks. kalder funktionen på følgende måde:

SendInteger(147);

skal følgende tegn sendes via UART 0: '1', '4' og '7'.

Hint:

Opret først i funktionen et lokalt array af "passende" størrelse.

Brug dernæst standard-funktionen **itoa()** til at konvertere "Tal" til en streng, der gemmes i dette array. Husk **#include <stdlib.h>**.

itoa(tal, array, 10) gemmer strengen svarende til "tal" i "array" (og 0-terminerer denne).

Brug derefter funktionen **SendString()** til at sende strengen.

Udvidelse af InitUART()

void InitUART(unsigned long BaudRate, unsigned char DataBit)

Skal initiere UART 0 til den ønskede BAUD-rate (300 - 115200) og det ønskede antal databits (5 - 8).

Hvis Parameteren BaudRate er mindre end 300 eller større end 115200, må der ikke ske nogen initiering af UART'en.

Hvis Parameteren DataBit er mindre end 5 eller større end 8, må der heller ikke ske nogen initiering af UART'en.

Vi antager, at Mega2560's clockfrekvens er 16 MHz.

Den værdi, der skal skrives til UBRR0, skal i funktionen beregnes på basis af "BaudRate"-parameteren og Mega2560's CPU clockfrekvens (afrunding kan forekomme).

Desuden skal UART'en initieres til:

- Asynkron mode.
- Ingen paritet.
- Både RX og TX enabled.
- 1 stop bit.
- Alle interrupts disabled.

Pointere og arrays

Brug af pointere

- Pointere **erklæres** via tegnet *

```
// Pointeren p er en pointer til en char  
char *p;  
// Pointeren fp er en pointer til en unsigned int  
unsigned int *fp;
```

- Adressen på en variabel kan tildeles en pointer via tegnet &

```
// a og b oprettes (2 char)  
char a,b;  
// Pointeren p peger nu på a  
p = &a;
```

- "Det som pointeren peger på" kan refereres via en *

```
// b = "Det, som p peger på" (=a)  
b = *p;
```

Pointere som parametre

```
void ombyt(int *a, int *b)
{
    // Midlertidig, lokal variabel
    int temp;
    // Gem "det som b peger på" i temp
    temp = *b;
    // Sæt "det som b peger på" lig med
    // "det som a peger på"
    *b = *a;
    // Sæt "det som a peger på" lig med temp
    *a = temp;
}

int tall, tal2, tal3;
ombyt(&tall, &tal2);
ombyt(&tal2, &tal3);
```

Erklæring af arrays

```
// Erklæring af arrayet tal, der indeholder 10 int (hver 16 bit)
// De har index fra 0 til 9
int tal[10];
// Erklæring af arrayet str, der indeholder 30 char (hver 8 bit)
// De har index fra 0 til 29
char str[30];
```

- Man kan også tildele elementerne en **startværdi**.

Hertil bruges tuborg-tegn {...}:

```
// Erklæring at arrayet tal_array, der indeholder 5 int (hver 16 bit)
// De enkelte elementer tildeles desuden en startværdi
int tal_array[5] = {12, 15, 27, 56, 74};
```

Navn på array = dens adresse

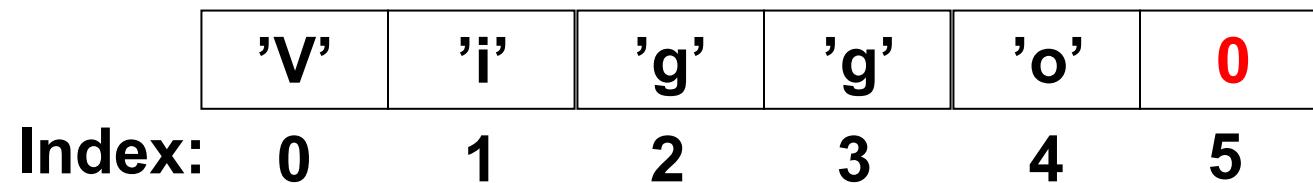
- Der findes en speciel C regel:
"Navnet på et array er adressen på array'ets første element (altså index 0)".
- Dette kan udnyttes, hvis vi ønsker at sætte en pointer til at pege på et arrays første element:

```
// Erklæring af arrayet str, ser indeholder 30 char (hver 8 bit)
// De har index fra 0 til 29
char str[30];

// En char pointer oprettes
char *p;

// str er navnet på et char array
// Vi sætter pointeren p til at pege på arrayet str
// OBS: Det vil ALTID være første element i arrayet
p = str;
// str[0] = 'A'
*p = 'A';
// Vi kan nu flytte pointeren en frem
// OBS: Det er pointeren, vi flytter - ikke det, den peger på
p++;
// str[1] = 'B'
*p = 'B';
```

Tekst-strenge



- En tekststrenge (**string**) er et **char array**, der indeholder tekst. Hvert element indeholder altså et tegn.
- Det sidste element skal være en **NUL-terminering** (værdien 0). Det fortæller os, hvor strengen slutter.
- I compileren kan vi skrive en streng direkte ved brug af anførselstegn: "...".
Compileren sætter automatisk NUL-termineringen på !

```
char str[6] = "Viggo";
```

Bemærk: ' bruges til en char, " bruges omkring char arrays!

Slut på lektion 17

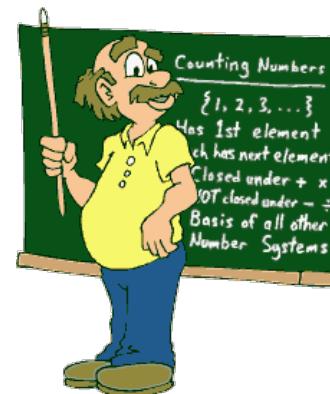
Kunne vi holde op med at skrige i bare fem minutter?!?! Jeg har en drønende hovedpine!!!



MSYS

Microcontroller Systems

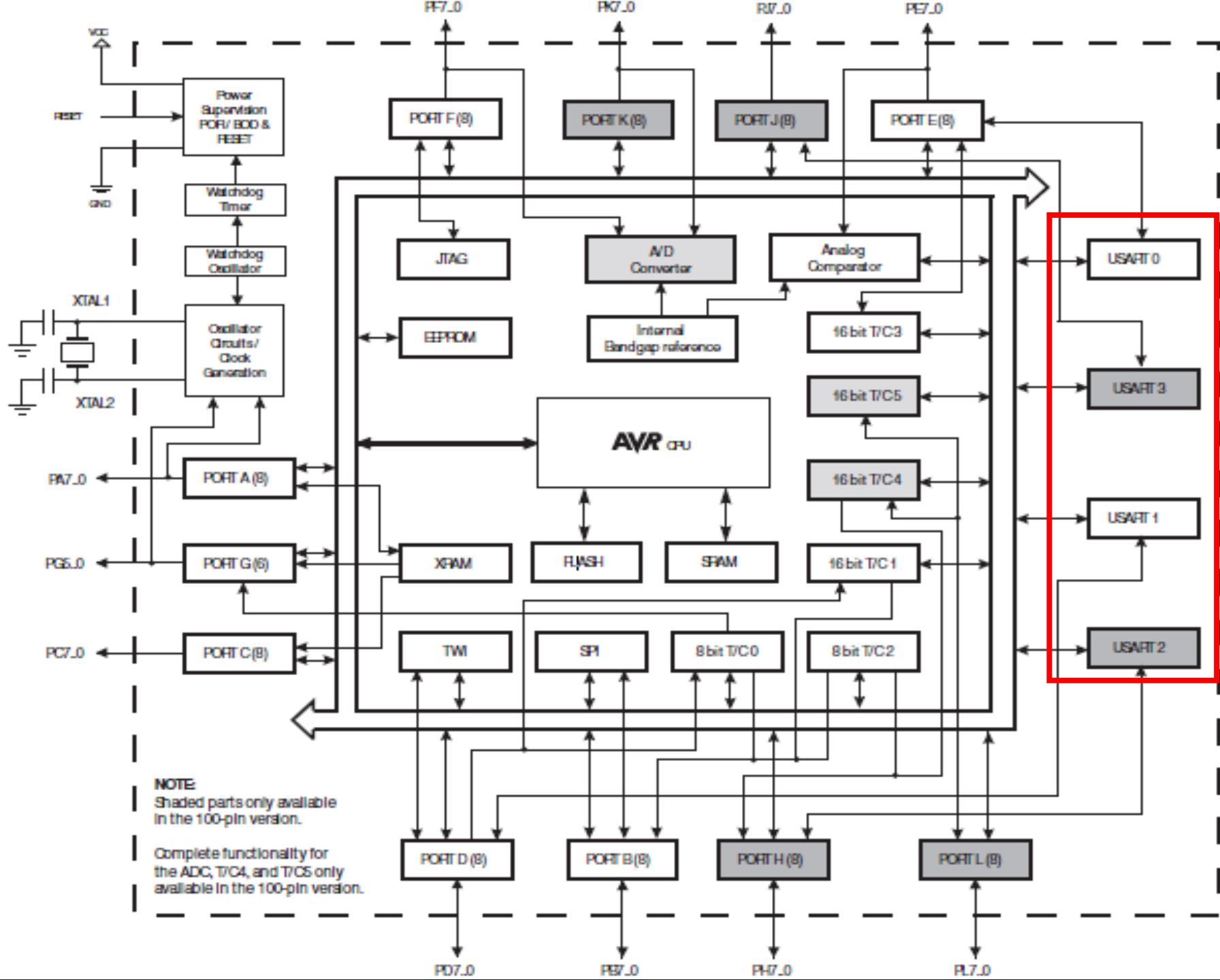
Lektion 18: Seriel kommunikation med interrupts m.m.



Slides from Henning Hargaard

Michael Alrøe

Mega2560 - 4 USART'er



Mega2560: USART pins

USART0 :

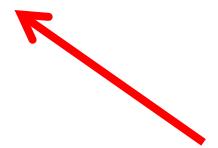
TXD0 = PE, ben 1

RXD0 = PE, ben 0

USART1 :

TXD1 = PD, ben 3

RXD1 = PD, ben 2



Via UART bridge til Arduino Mega2560
USB - stikket

USART2 :

TXD2 = PH, ben 1

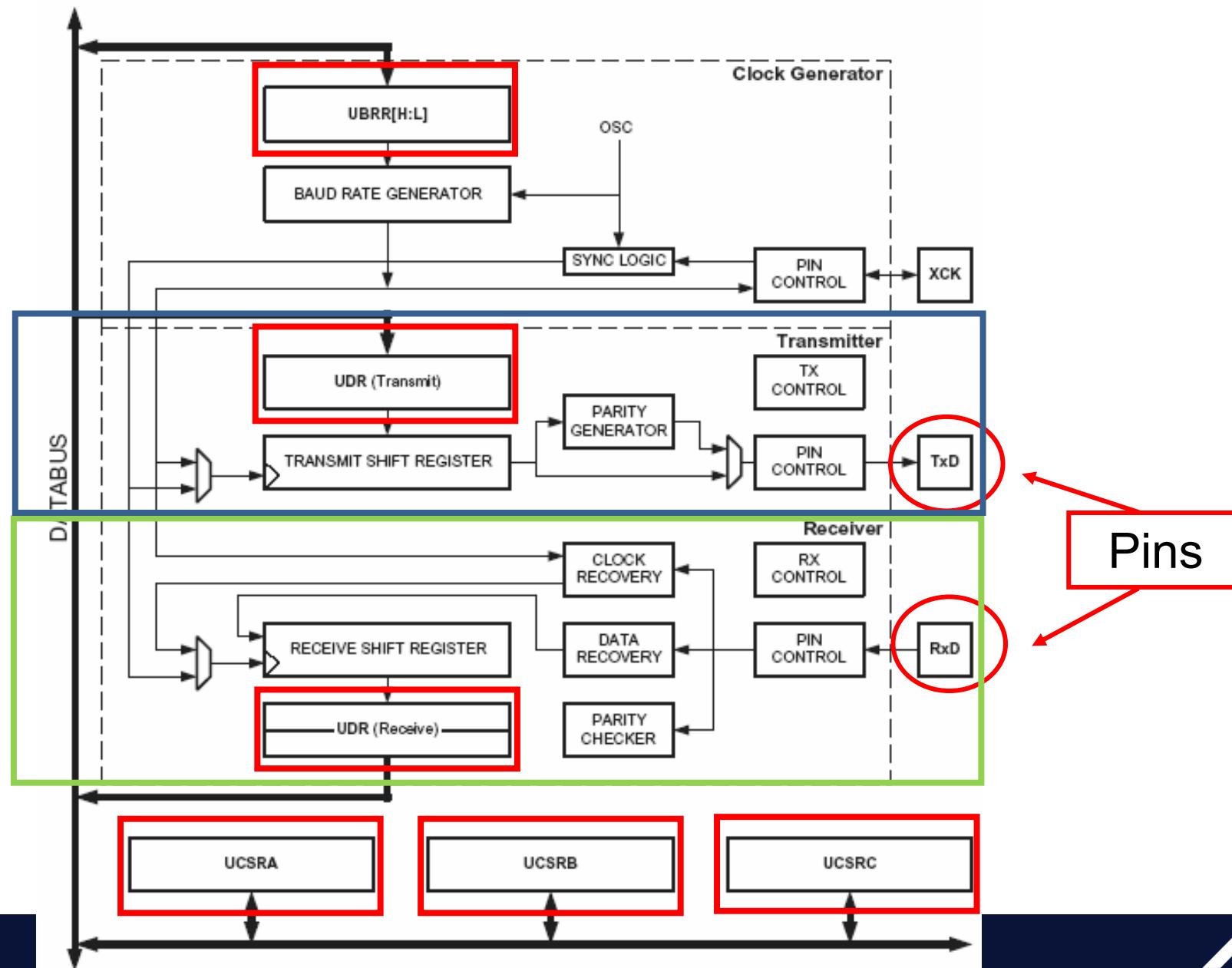
RXD2 = PH, ben 0

USART3 :

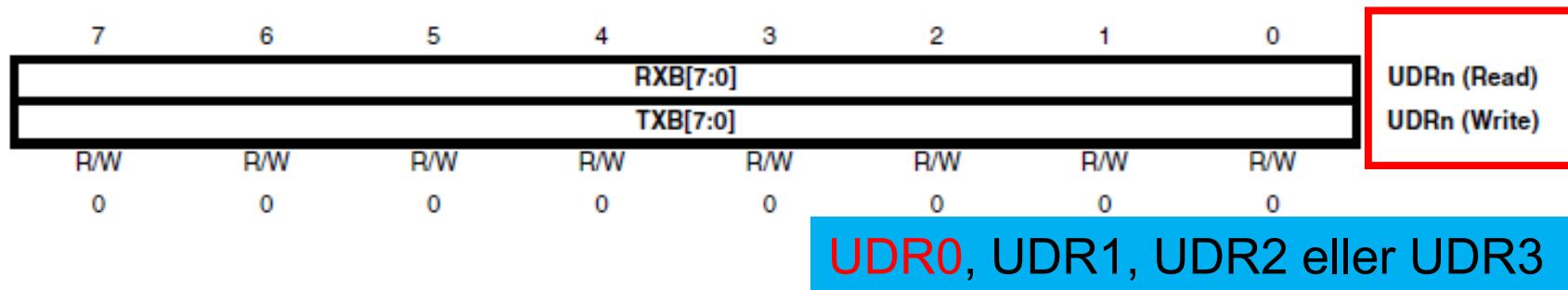
TXD3 = PJ, ben 1

RXD3 = PJ, ben 0

Mega32/Mega2560 USART



Mega2560: UDRn. Usart Data Registers



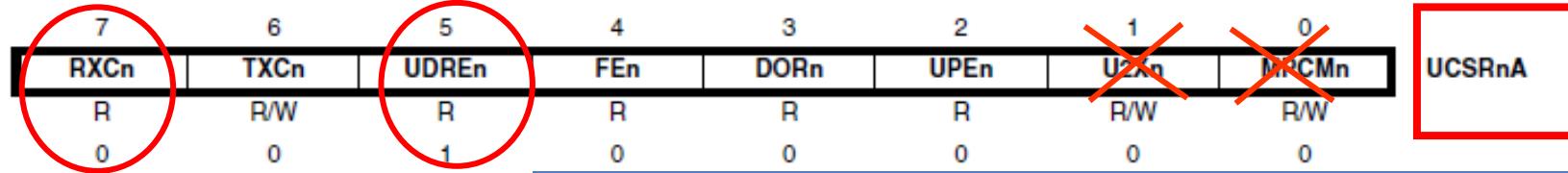
Bemærk: Fysisk to forskellige registre med samme navn (UDRn)

char x;

Læsning af modtaget tegn : **x = UDR0;**

Send tegn : **UDR0 = x;**

Mega2560: UCSRnA: Control and Status Registers A



UCSR0A, UCSR1A, UCSR2A eller UCSR3A

Når RXCn er 1: Nyt tegn modtaget ("kan hentes i UDRn").

Når TXC er 1: "Sender tom" (klar til at sende nyt tegn og sendeskifteredisteret tomt).

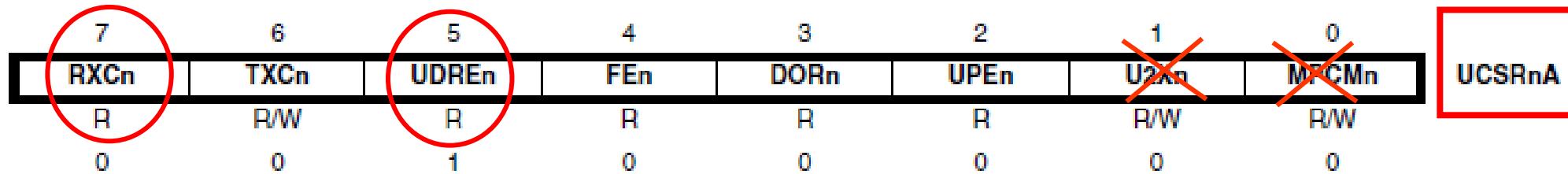
Når UDREn er 1: Klar til at sende nyt tegn ("der må skrives til UDRn").

FEn : "Framing Error" (modtaget tegn har fejl i stop bit).

DORn : "Data overrun" (tegn modtaget, inden foregående er blevet læst af SW).

PEn : "Parity Error" (modtaget tegn har paritetsfejl).

Mega2560: UCSRnA: Control and Status Registers A



UCSR0A, UCSR1A, UCSR2A eller UCSR3A

RXCn -> 1: Nyt tegn modtaget ("kan hentes i UDR").

TXC -> 1: "Sender tom" (klar til at sende nyt tegn og sende skifteredisteret tomt).

UDREn ->1: Klar til at sende nyt tegn ("der må skrives til UDR").

FEn -> 1: "Framing Error" (modtaget tegn har fejl i stop bit).

DORn -> 1: "Data overrun" (tegn modtaget, inden foregående er blevet læst af SW).

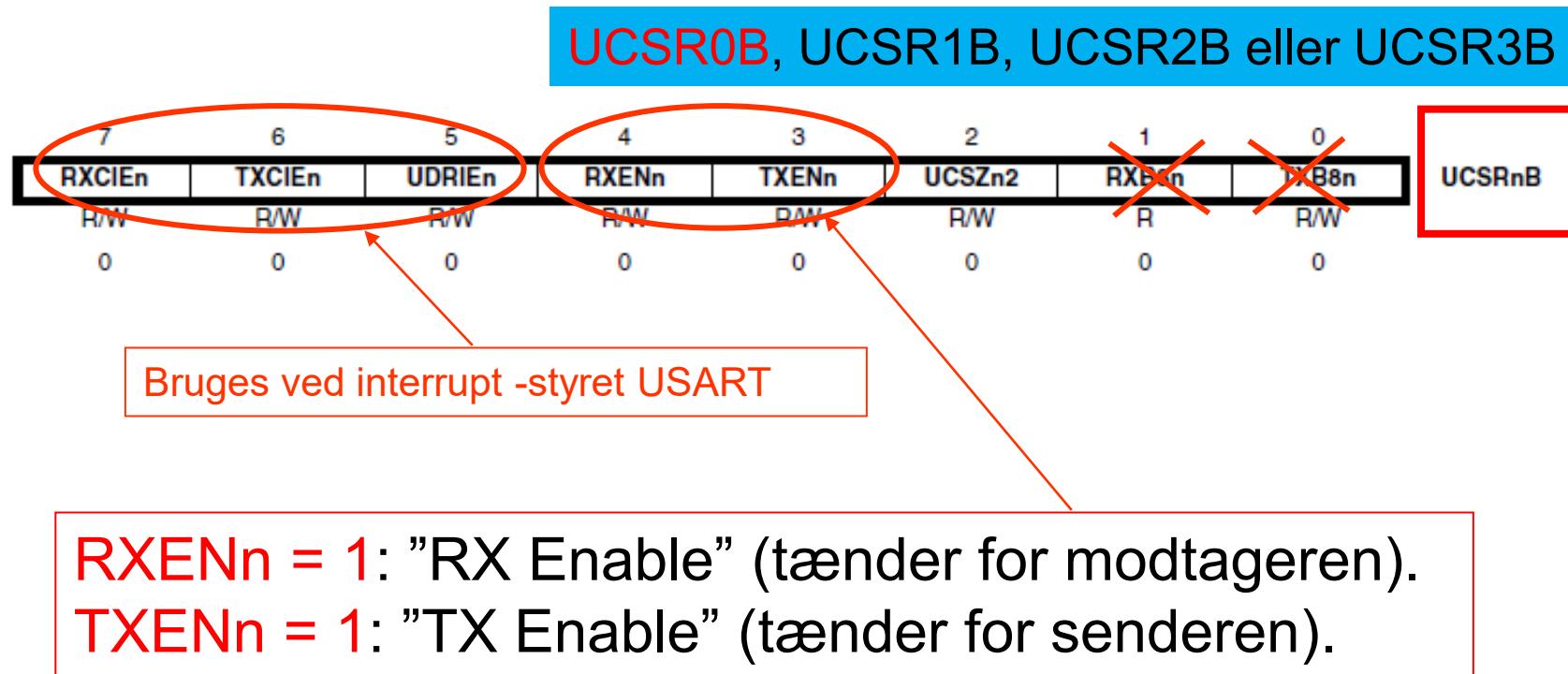
PEn -> 1: "Parity Error" (modtaget tegn har paritetsfejl).

Test ("socrative.com", Room = AUQUIZ)

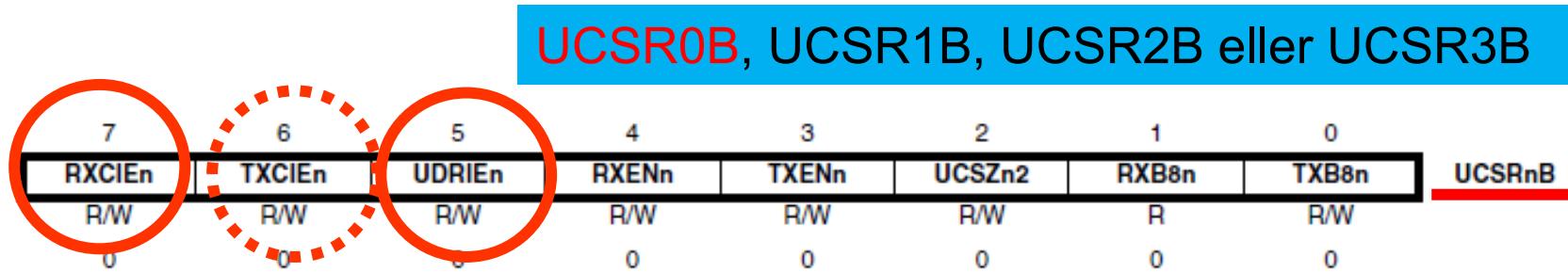
- Hvilken metode er korrekt for at vente på, at et nyt tegn modtages (når der ikke anvendes interrupt) ?
- A:
if (UCSR0A & 0b10000000) != 0)
return UDR0;
- B:
while (UCSR0A & 0b10000000 == 0)
{ }
return UDR0;
- C:
while (UCSR0A & 0b10000000 != 0)
{ }
return UDR0;



Mega2560: UCSRnB. Control and Status Register B



Mega2560: USART interrupt enables



RXCIEn = 1: USART Rx Interrupt Enable.

Interrupt hver gang et **nyt tegn** er modtaget.

Tegnet vil ligge klar i UDR, når vi får interruptet.

(TXCIEn = 1: USART Tx Interrupt Enable).

Interrupt hver gang **sende-skifтерegisteret** bliver tomt
og UDR klar til at sende et nyt tegn.

Betyder, at man må sende næste tegn (til UDR).

(Oftest benyttes det andet sender-interrupt i stedet).

UDRIEn = 1: USART Data Register Empty Interrupt Enable.

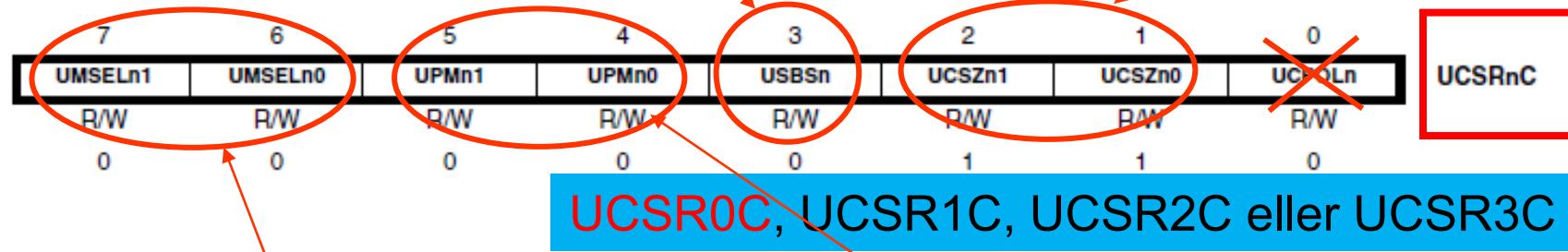
Interrupt hver gang UDR er klar til at sende et tegn.

Betyder, at man kan sende næste tegn (til UDR).

Mega2560: UCSRnC. Control and Status Register C

UCSZn1 og UCSZn0 : Antal data bits (se næste side).

USBSn : 0 = 1 stop bit. 1 = 2 stop bits.



UCSR0C, UCSR1C, UCSR2C eller UCSR3C

UMSELn1 = 0 og UMSELn0 = 0 :
Asynkron mode !

UPMn1 og UPMn0: Valg af paritet:

UPMn1	UPMn0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

Test ("socrative.com", Room = AUQUIZ)

- Vi antager, at der benyttes 8 databit og EVEN parity. Hvilken bitstrøm har paritetsfejl ?

A:

Data = 11101001 Paritetsbit = 1

B:

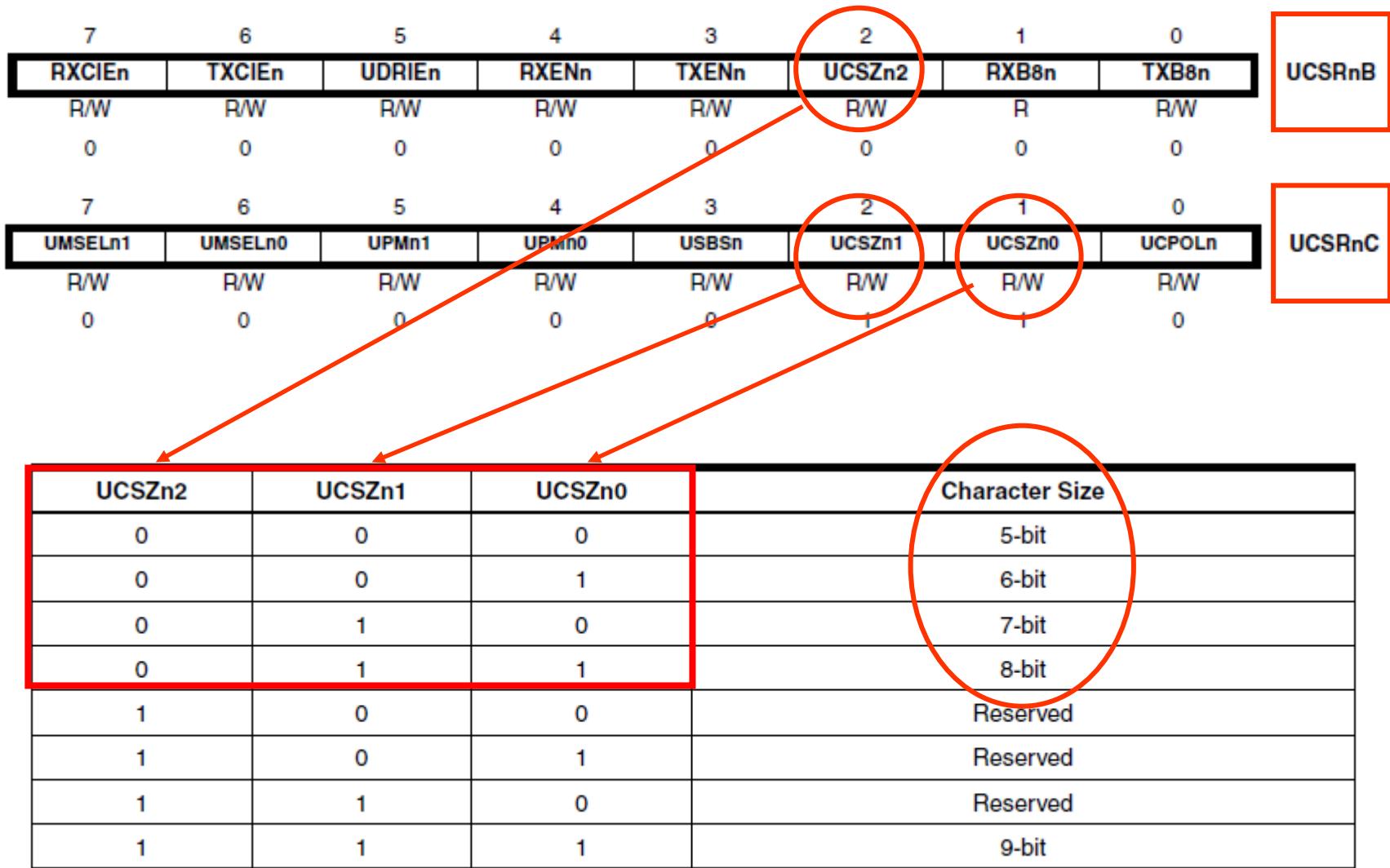
Data = 10001011 Paritetsbit = 0

C:

Data = 10001011 Paritetsbit = 1

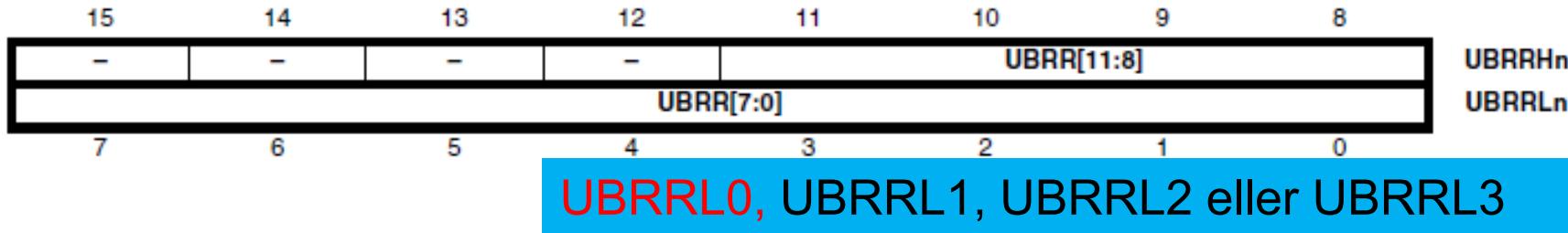


Mega2560: Antal data bits



Mega2560: UBRRHn + UBRLLn. Baud Rate Registre

UBRRH0, UBRRH1, UBRRH2 eller UBRRH3



Operating Mode	Equation for Calculating Baud Rate ⁽¹⁾	Equation for Calculating UBRR Value
Asynchronous Normal mode (U2Xn = 0)	$BAUD = \frac{f_{OSC}}{16(UBRRn + 1)}$	$UBRRn = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed mode (U2Xn = 1)	$BAUD = \frac{f_{OSC}}{8(UBRRn + 1)}$	$UBRRn = \frac{f_{OSC}}{8BAUD} - 1$

"UBRRn" = (256 * UBRRHn) + UBRLLn

Test ("socrative.com", Room = AUQUIZ)

- CPU clock frekvens = 3,6864 MHz
UBRRH = 2
UBRRL = 255
Hvilken BAUD rate anvendes ?

A:

300 bit/s

B:

1200 bit/s

C:

9600 bit/s

- D:

115200 bit/s



Mega2560: USART Interrupt vektorer (1 af 2)

Vector No.	Program Address ⁽¹⁾	Source	Interrupt Definition
1	\$0000 ⁽¹⁾	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	\$0002	INT0	External Interrupt Request 0
3	\$0004	INT1	External Interrupt Request 1
4	\$0006	INT2	External Interrupt Request 2
5	\$0008	INT3	External Interrupt Request 3
6	\$000A	INT4	External Interrupt Request 4
7	\$000C	INT5	External Interrupt Request 5
8	\$000E	INT6	External Interrupt Request 6
9	\$0010	INT7	External Interrupt Request 7
10	\$0012	PCINT0	Pin Change Interrupt Request 0
11	\$0014	PCINT1	Pin Change Interrupt Request 1
12	\$0016 ⁽²⁾	PCINT2	Pin Change Interrupt Request 2
13	\$0018	WDT	Watchdog Time-out Interrupt
14	\$001A	TIMER2 COMPA	Timer/Counter2 Compare Match A
15	\$001C	TIMER2 COMPB	Timer/Counter2 Compare Match B
16	\$001E	TIMER2 OVF	Timer/Counter2 Overflow
17	\$0020	TIMER1 CAPT	Timer/Counter1 Capture Event
18	\$0022	TIMER1 COMPA	Timer/Counter1 Compare Match A
19	\$0024	TIMER1 COMPB	Timer/Counter1 Compare Match B
20	\$0026	TIMER1 COMPC	Timer/Counter1 Compare Match C
21	\$0028	TIMER1 OVF	Timer/Counter1 Overflow
22	\$002A	TIMER0 COMPA	Timer/Counter0 Compare Match A
23	\$002C	TIMER0 COMPB	Timer/Counter0 Compare match B
24	\$002E	TIMER0 OVF	Timer/Counter0 Overflow
25	\$0030	SPI, STC	SPI Serial Transfer Complete
	26	USART0 RX	USART0 Rx Complete
	27	USART0 UDRE	USART0 Data Register Empty
	28	USART0 TX	USART0 Tx Complete
29	\$0038	ANALOG COMP	Analog Comparator
30	\$003A	ADC	ADC Conversion Complete

Mega2560: USART Interrupt vektorer (2 af 2)

Vector No.	Program Address ⁽²⁾	Source	Interrupt Definition
31	\$003C	EE READY	EEPROM Ready
32	\$003E	TIMER3 CAPT	Timer/Counter3 Capture Event
33	\$0040	TIMER3 COMPA	Timer/Counter3 Compare Match A
34	\$0042	TIMER3 COMPB	Timer/Counter3 Compare Match B
35	\$0044	TIMER3 COMPC	Timer/Counter3 Compare Match C
36	\$0046	TIMER3 OVF	Timer/Counter3 Overflow
37	\$0048	USART1 RX	USART1 Rx Complete
38	\$004A	USART1 UDRE	USART1 Data Register Empty
39	\$004C	USART1 TX	USART1 Tx Complete
40	\$004E	TWI	2-wire Serial Interface
41	\$0050	SPM READY	Store Program Memory Ready
42	\$0052 ⁽³⁾	TIMER4 CAPT	Timer/Counter4 Capture Event
43	\$0054	TIMER4 COMPA	Timer/Counter4 Compare Match A
44	\$0056	TIMER4 COMPB	Timer/Counter4 Compare Match B
45	\$0058	TIMER4 COMPC	Timer/Counter4 Compare Match C
46	\$005A	TIMER4 OVF	Timer/Counter4 Overflow
47	\$005C ⁽³⁾	TIMER5 CAPT	Timer/Counter5 Capture Event
48	\$005E	TIMER5 COMPA	Timer/Counter5 Compare Match A
49	\$0060	TIMER5 COMPB	Timer/Counter5 Compare Match B
50	\$0062	TIMER5 COMPC	Timer/Counter5 Compare Match C
51	\$0064	TIMER5 OVF	Timer/Counter5 Overflow
52	\$0066 ⁽³⁾	USART2 RX	USART2 Rx Complete
53	\$0068 ⁽³⁾	USART2 UDRE	USART2 Data Register Empty
54	\$006A ⁽³⁾	USART2 TX	USART2 Tx Complete
55	\$006C ⁽³⁾	USART3 RX	USART3 Rx Complete
56	\$006E ⁽³⁾	USART3 UDRE	USART3 Data Register Empty
57	\$0070 ⁽³⁾	USART3 TX	USART3 Tx Complete

Mega2560: USART0 - ISR navne

```
ISR (USART0_RX_vect)
{
}
```

```
| ISR (USART0_TX_vect)
| {
| }
```

```
ISR (USART0_UDRE_vect)
{
}
```

Mega2560: ISR navne (USART0 - 3)

USART 0 interrupts	ISR name
RX	USART0_RX_vect
TX	USART0_TX_vect
UDR Empty	USART0_UDRE_vect

USART 2 interrupts	ISR name
RX	USART2_RX_vect
TX	USART2_TX_vect
UDR Empty	USART2_UDRE_vect

USART 1 interrupts	ISR name
RX	USART1_RX_vect
TX	USART1_TX_vect
UDR Empty	USART1_UDRE_vect

USART 3 interrupts	ISR name
RX	USART3_RX_vect
TX	USART3_TX_vect
UDR Empty	USART3_UDRE_vect

Metode for interruptstyret USART0 modtager

```
ISR (USART0_RX_vect)
{
    char x;
    // Hent modtaget tegn
    x = UDR0;
    // - og brug så x til "noget"
    .....
}
```

Bemærk, at vi ikke behøver at "vente på, at der modtages et nyt tegn".

Vi får interruptet, fordi et tegn allerede er modtaget !

LAB11: UART driver

```
*****  
* "uart.h":  
* Header file for Mega2560 UART driver.  
* Using UART 0.  
* Henning Hargaard, 8/4 2019  
*****/  
  
void InitUART(unsigned long BaudRate, unsigned char DataBit);  
unsigned char CharReady();  
char ReadChar();  
void SendChar(char Tegn);  
void SendString(char* Streng);  
void SendInteger(int Tal);  
*****/
```

Driverens header fil.

InitUART()

void InitUART(unsigned long BaudRate, unsigned char DataBit)

Skal initiere UART 0 til den ønskede BAUD-rate (300 - 115200) og det ønskede antal databits (5 - 8).

Hvis Parameteren BaudRate er mindre end 300 eller større end 115200, må der ikke ske nogen initiering af UART'en.

Hvis Parameteren DataBit er mindre end 5 eller større end 8, må der heller ikke ske nogen initiering af UART'en.

Vi antager, at Mega2560's clockfrekvens er 16 MHz.

Den værdi, der skal skrives til UBRR0, skal i funktionen beregnes på basis af "BaudRate"-parameteren og Mega2560's CPU clockfrekvens (afrunding kan forekomme).

Desuden skal UART'en initieres til:

- Asynkron mode.
- Ingen paritet.
- Både RX og TX enabled.
- 1 stop bit.
- Alle interrupts disabled.

Løsning: InitUART()

```
// Constants
#define XTAL 16000000

void InitUART(unsigned long BaudRate, unsigned char DataBit)
{
    if ((BaudRate >= 300) && (BaudRate <= 115200) && (DataBit >=5) && (DataBit <= 8))
    {
        // No interrupts enabled
        // Receiver enabled
        // Transmitter enabled
        // No 9 bit operation
        UCSR0B = 0b00011000;
        // Asynchronous operation, 1 stop bit
        // Bit 2 and bit 1 controls the number of data bits
        UCSR0C = (DataBit-5)<<1;
        // Set Baud Rate according to the parameter BaudRate
        UBRR0 = XTAL/(16*BaudRate) - 1;
    }
}
```

Karakter funktioner

unsigned char CharReady()

Meddeler, om UART 0 har modtaget et tegn.

Hvis et tegn er modtaget, returneres en værdi forskellig fra 0 (= TRUE).

Hvis der ikke er modtaget et tegn, returneres værdien 0 (= FALSE).

Funktionen skal ikke afvente modtagelse af et tegn, men blot returnere oplysningen om, hvorvidt et tegn er modtaget.

char ReadChar()

Returnerer et modtaget tegn fra UART 0's modtageregister (UDR).

Funktionen skal først afvente, at et tegn modtages (bit RXC0 i registeret UCSRA0).

Derefter skal tegnet i UDR0 returneres.

void SendChar(char Tegn)

Sender et tegn via UART 0. Tegnets overføres som parameter.

Inden tegnet skrives til data registeret (UDR0), skal funktionen afvente "UART data register empty" (bit UDRE0 i registeret UCSRA0).

SendString()

void SendString(char* Streng)

Udskriver en 0-termineret tekststreng ved hjælp af UART 0.

Funktion modtager som parameter en pointer til den streng, som vi ønsker udskrevet.

Pointeren peger altid på det første tegn i strengen, som altså er 0-termineret.

Brugeren har på forhånd (altså inden denne funktion kaldes) oprettet og lagret strengen.

Nedenstående viser i pseudo-kode, hvordan funktionen kan implementeres:

```
while ("Det som pointeren peger på" ikke er 0)
{
    SendChar("Det som pointeren peger på");
    Flyt pointeren en plads frem;
}
```

Brug af pointere

- Pointere **erklæres** via tegnet *

```
// Pointeren p er en pointer til en char  
char *p;  
// Pointeren fp er en pointer til en unsigned int  
unsigned int *fp;
```

- Adressen på en variabel kan tildeles en pointer via tegnet &

```
// a og b oprettes (2 char)  
char a,b;  
// Pointeren p peger nu på a  
p = &a;
```

- "Det som pointeren peger på" kan refereres via en *

```
// b = "Det, som p peger på" (=a)  
b = *p;
```

SendInteger()

void SendInteger(int Tal)

Denne funktion skal udskrive værdien af integer "Tal", der modtages som parameter.

Hvis man f.eks. kalder funktionen på følgende måde:

SendInteger(147);

skal følgende tegn sendes via UART 0: '1', '4' og '7'.

Hint:

Opret først i funktionen et lokalt array af "passende" størrelse.

Brug dernæst standard-funktionen **itoa()** til at konvertere "Tal" til en streng, der gemmes i dette array. Husk **#include <stdlib.h>**.

itoa(tal, array, 10) gemmer strengen svarende til "tal" i "array" (og 0-terminerer denne).

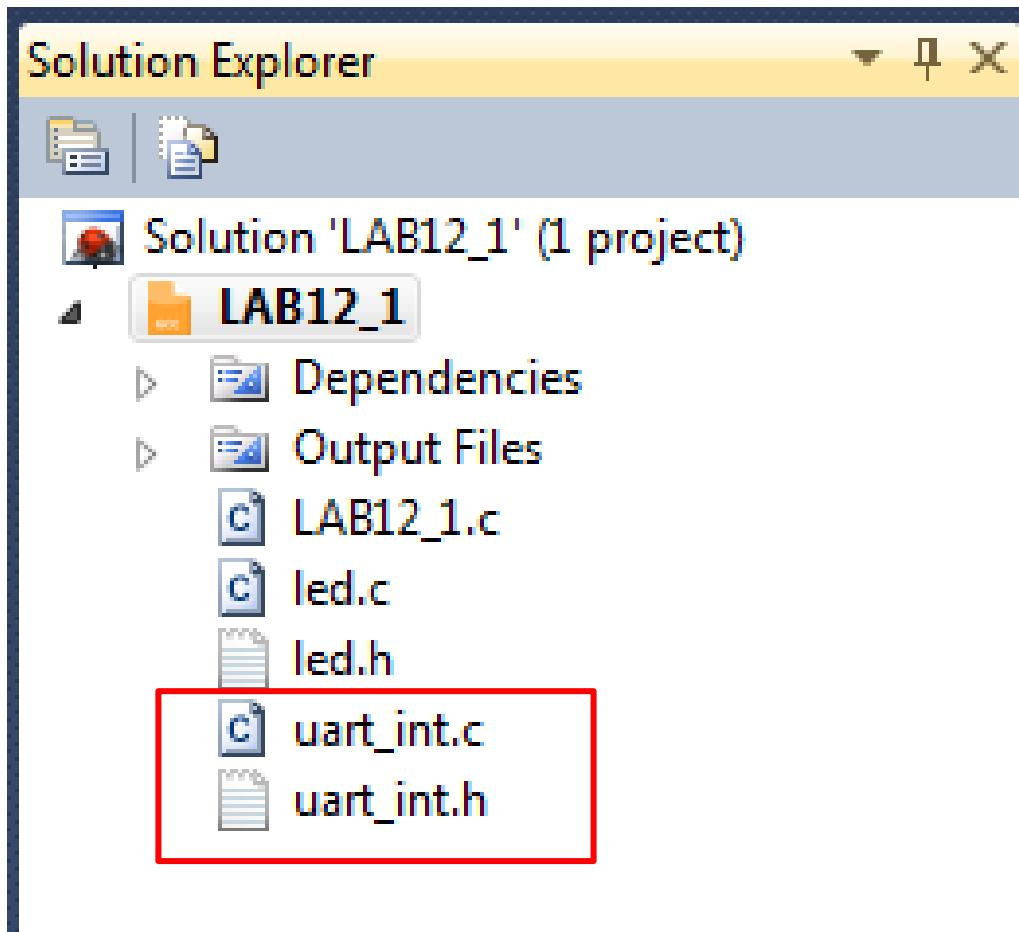
Brug derefter funktionen **SendString()** til at sende strengen.

LAB12: Mulighed for UART interrupt

```
*****
* "uart_int.h": *
* Header file for Mega2560 UART driver. *
* Using UART 0. *
* If parameter Rx_Int <> 0 : *
* Receiver interrupt will be enabled *
* *
* Henning Hargaard, 10/4 2019 *
*****
void InitUART(unsigned long BaudRate, unsigned char DataBit, unsigned char Rx_Int);
unsigned char CharReady();
char ReadChar();
void SendChar(char Tegn);
void SendString(char* Streng);
void SendInteger(int Tal);
*****
```

Hvis RX_int er 0, skal UART modtage-interruptet være disabled (som i ”den gamle driver”).
Hvis RX_int er forskellig fra 0, skal UART modtage-interruptet være enabled.

LAB12, del 1a



LAB12, del 1a

Skriv i "LAB12.c"'s main()-funktion kode, der:

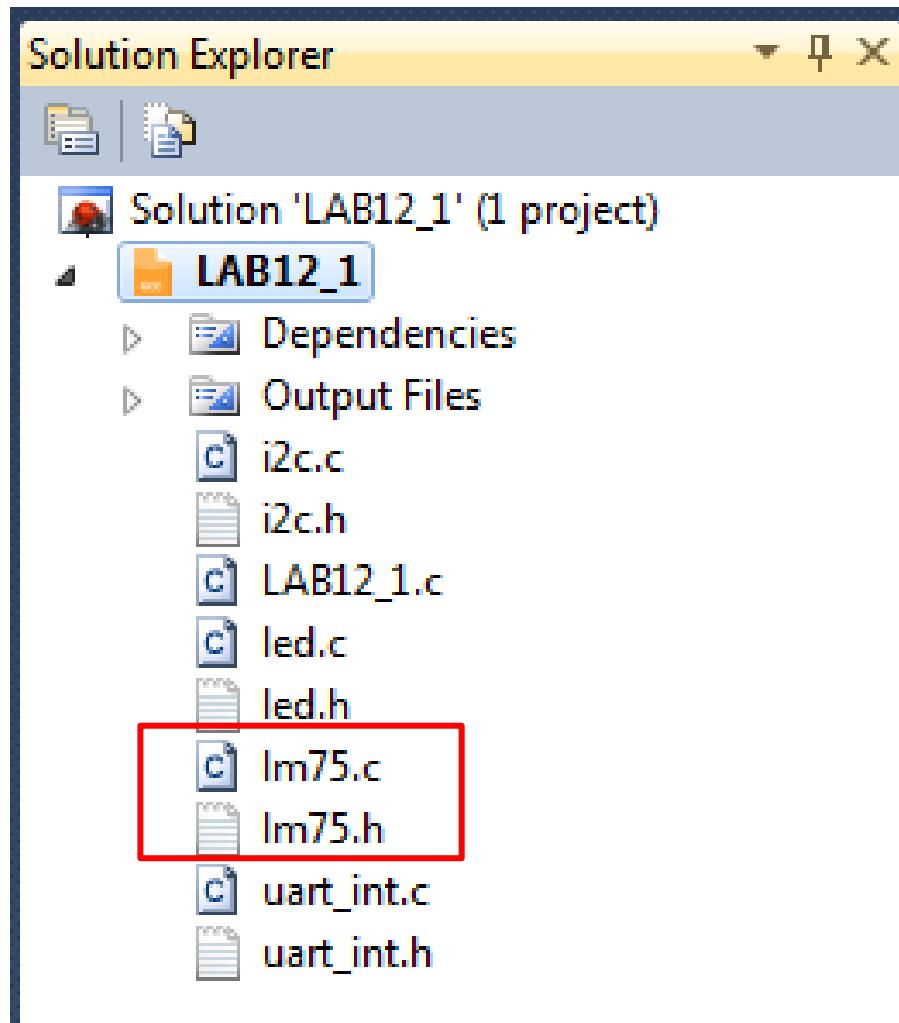
- Initierer UART'en, således at UART modtage-interruptet bliver enabled.
- Foretager global interrupt enable.
- Initierer LED driveren.
- Herefter går i en uendelig sløjfe (laver "ingenting").

Skriv også i "LAB12.c" en interrupt service rutine (ISR) for UART modtage interrupts.

I denne ISR skal følgende ske:

- Aflæs UART'ens modtageregister UDR (= modtaget tegn) til en lokal variabel.
- Hvis det modtagne tegn er '0', '1', '2', '3', '4', '5', '6' eller '7' skal den tilsvarende lysdiode toggles (hvis f.eks. tegnet er '3' skal lysdiode nummer 3 toggles). Derefter sendes der en besked retur til terminalen: "LED nummer x er toggled". "x" er lysdiodens nummer.

LAB12, del 1b



LAB12, del 1b



LAB12, del 1b

```
*****  
* "LM75.h":  
* Header file for LM75 driver.  
* LM75 is an I2C temperature sensor.  
* Temperature is returned in HALFs of  
* centigrades.  
*  
* Henning Hargaard, 13/11 2015  
*****/  
void LM75_init();  
int LM75_temperature(unsigned char SensorAddress)  
*****/
```

LAB12, del 1b

Foretag disse tilføjelser til "LAB12_1.c" :

1. #include "LM75.h"
2. Inden while(1) – sløjfen: Kald LM75_init().
3. I while(1) – sløjfen:

Skriv kode, der kalder LM75_temperature(0) og gemmer det returnerede i en variabel.

Parameteren "SensorAddress" skal altså være 0.

LM75_temperature(0) vil returnere LM75's temperatur i enheden "halve grader celcius".

Hvis temperaturen f.eks. er 24,5 grader, vil heltallet 49 blive returneret.

Udskriv temperaturen til terminalen i formatet "24,5 grader".

Brug hertil relevante funktioner fra UART-driveren.

Hold en pause på 1 sekund (brug _delay_ms(1000)) efter hver aflæsning/udskrivning.

LAB12, del 1b

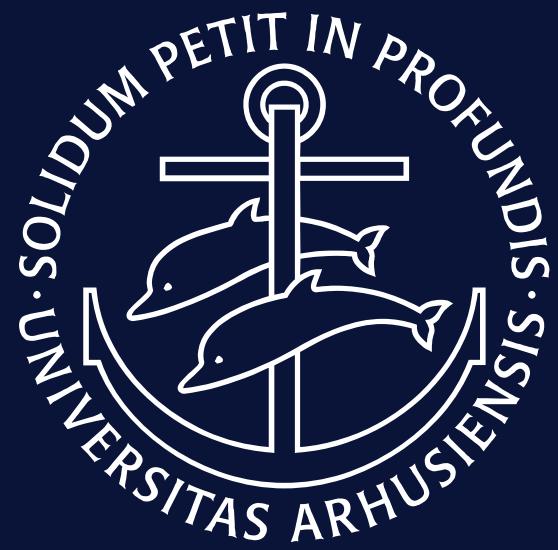
Connect COM2 Baud: 9600 ASCII

Receive

```
LED nummer 2 er toggled  
25,5 grader  
LED nummer 3 er toggled  
25,0 grader  
25,0 grader  
25,0 grader  
LED nummer 2 er toggled  
25,0 grader  
LED nummer 2 er toggled  
24,5 grader  
LED nummer 3 er toggled  
24,5 grader  
LED nummer 2 er toggled  
24,5 grader  
24,5 grader  
24,5 grader
```

Slut på lektion 18

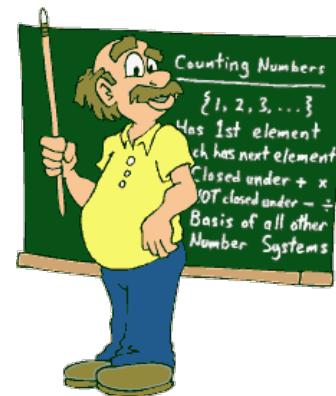




MSYS

Microcontroller Systems

Lektion 19: A/D konvertering



Slides from Henning Hargaard

Michael Alrøe

Hvorfor ADC?

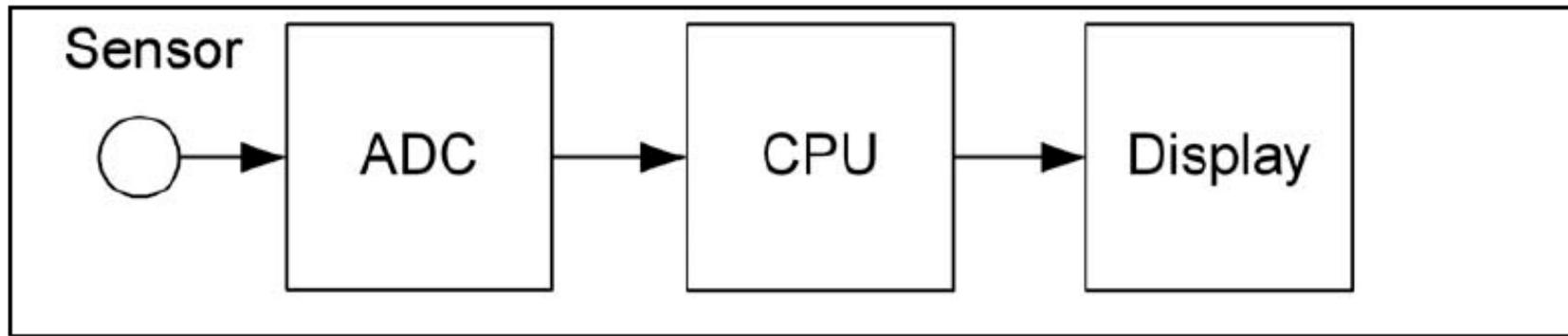
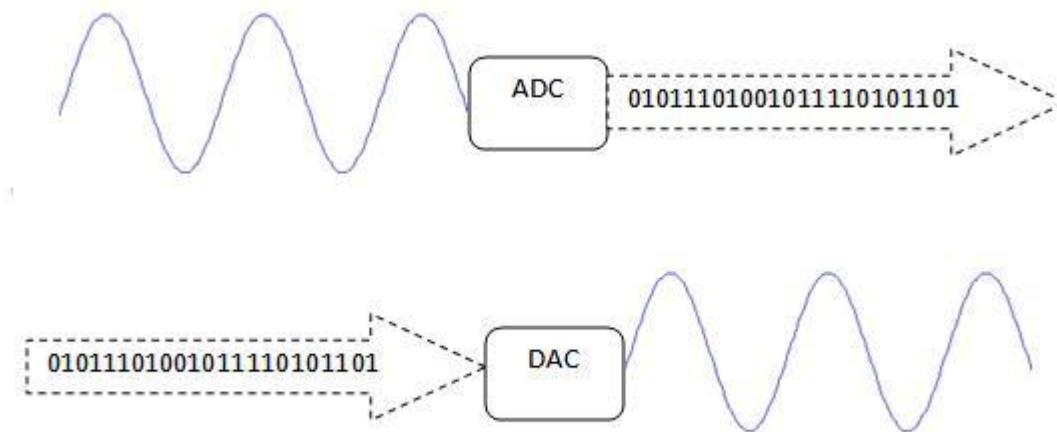
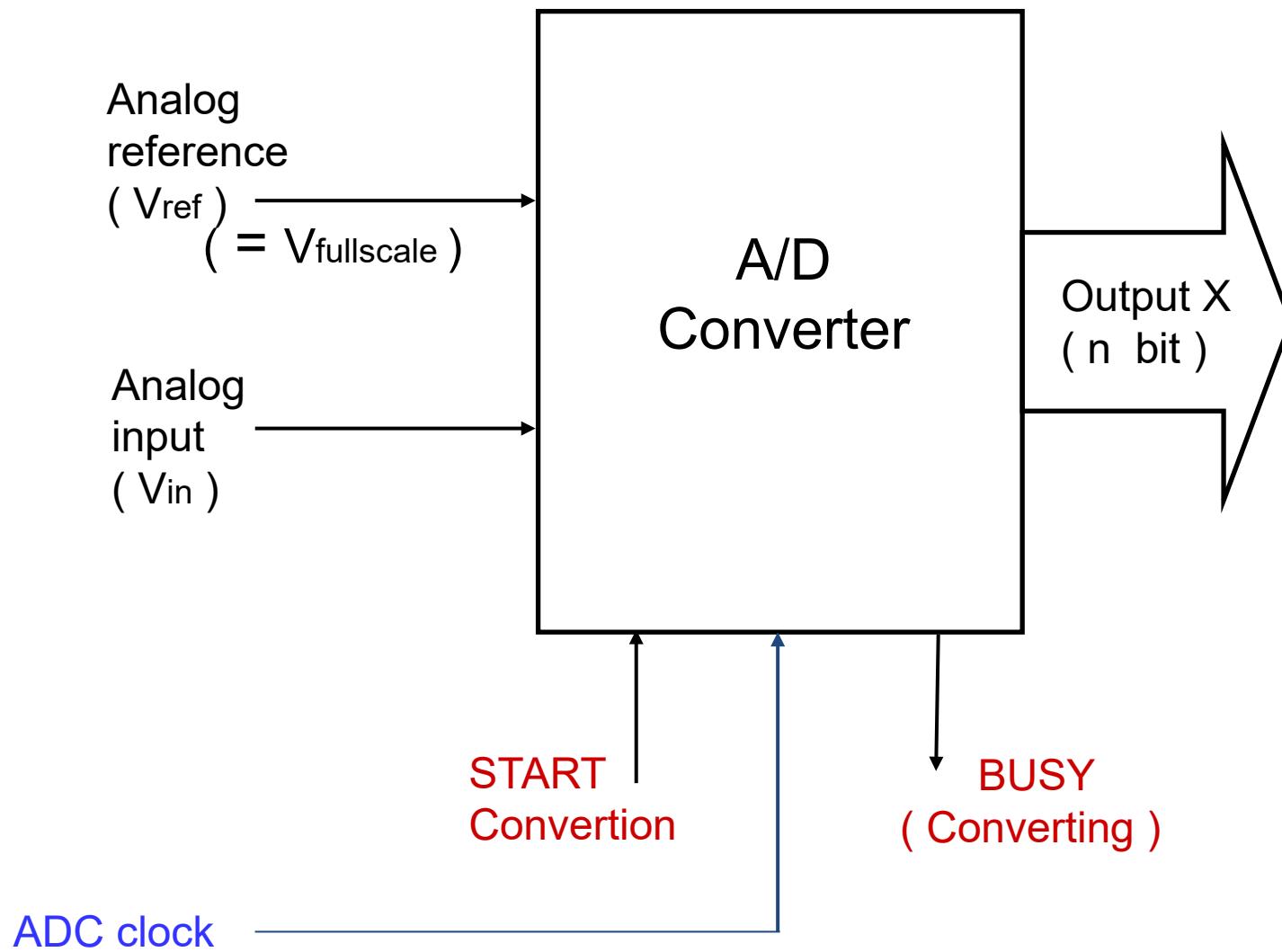


Figure 13-1. Microcontroller Connection to Sensor via ADC



ADC generelt



ADC generelt

Generelle formler:

$$V_{res} = \frac{V_{ref}}{2^N} = 1 \text{ LSB}$$

$$\frac{V_{in}}{V_{ref}} = \frac{ADC_{result}}{(2^N - 1)}$$

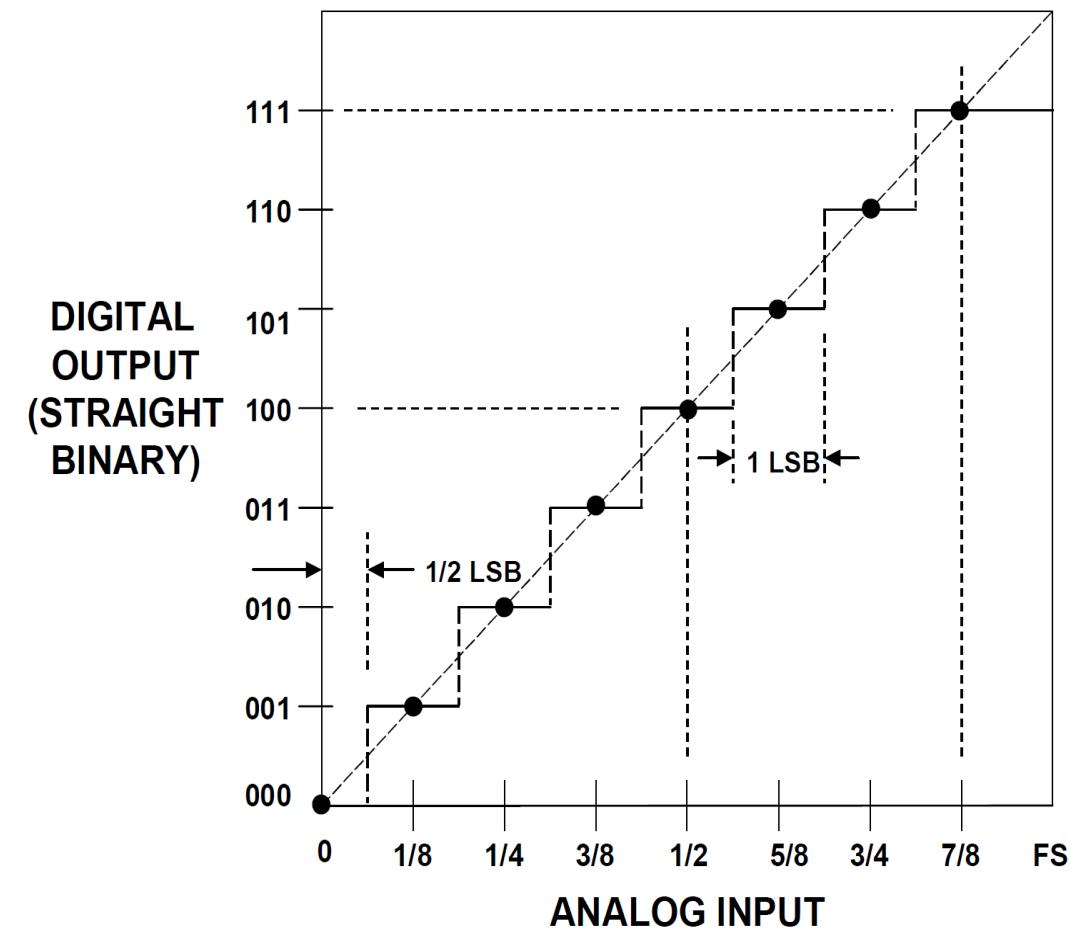


Figure 2.5: Transfer Function for Ideal Unipolar 3-bit ADC

Test ("socrative.com"): Room = AUQUIZ

En A/D konverters referencespænding er 5 volt,
og måleresultatet repræsenteres med 12 bit.

Hvad er A/D konverterens måleopløsning ?

- A: 1,22 mV
- B: 12 mV
- C: 4,88 mV
- D: 5,12 mV

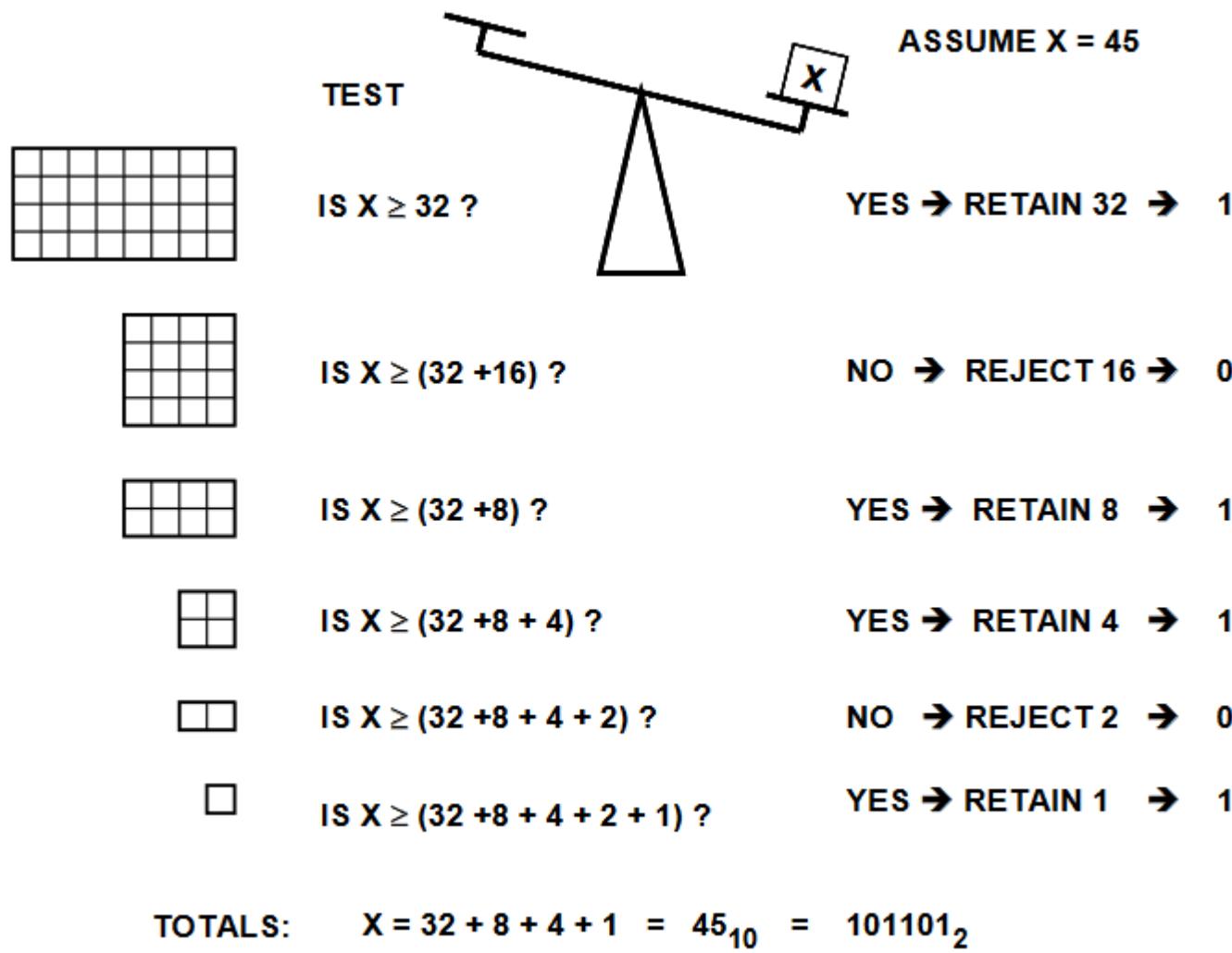


Mega32/Mega2560 ADC

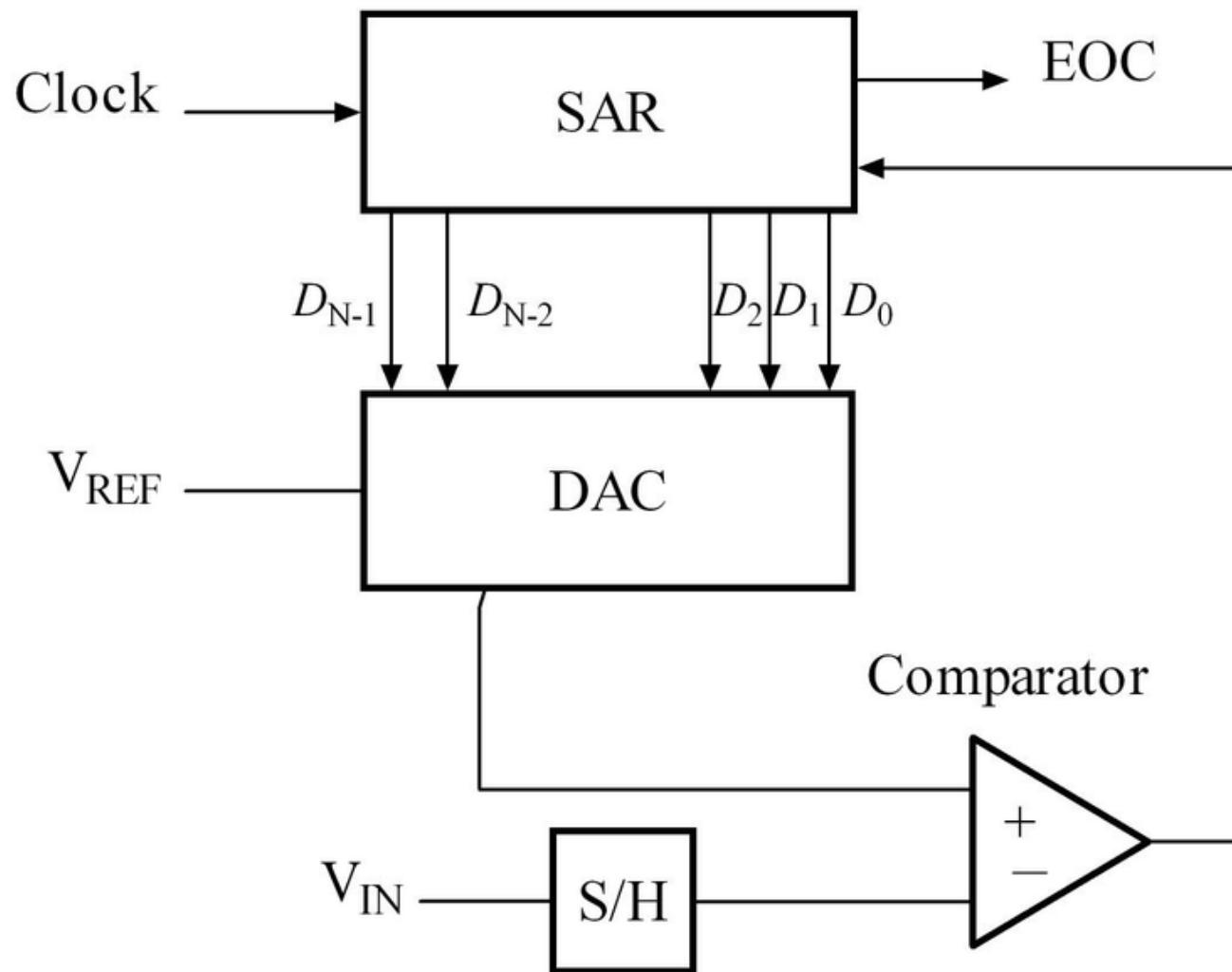


AVR's ADC = "Successiv Approximation ADC"

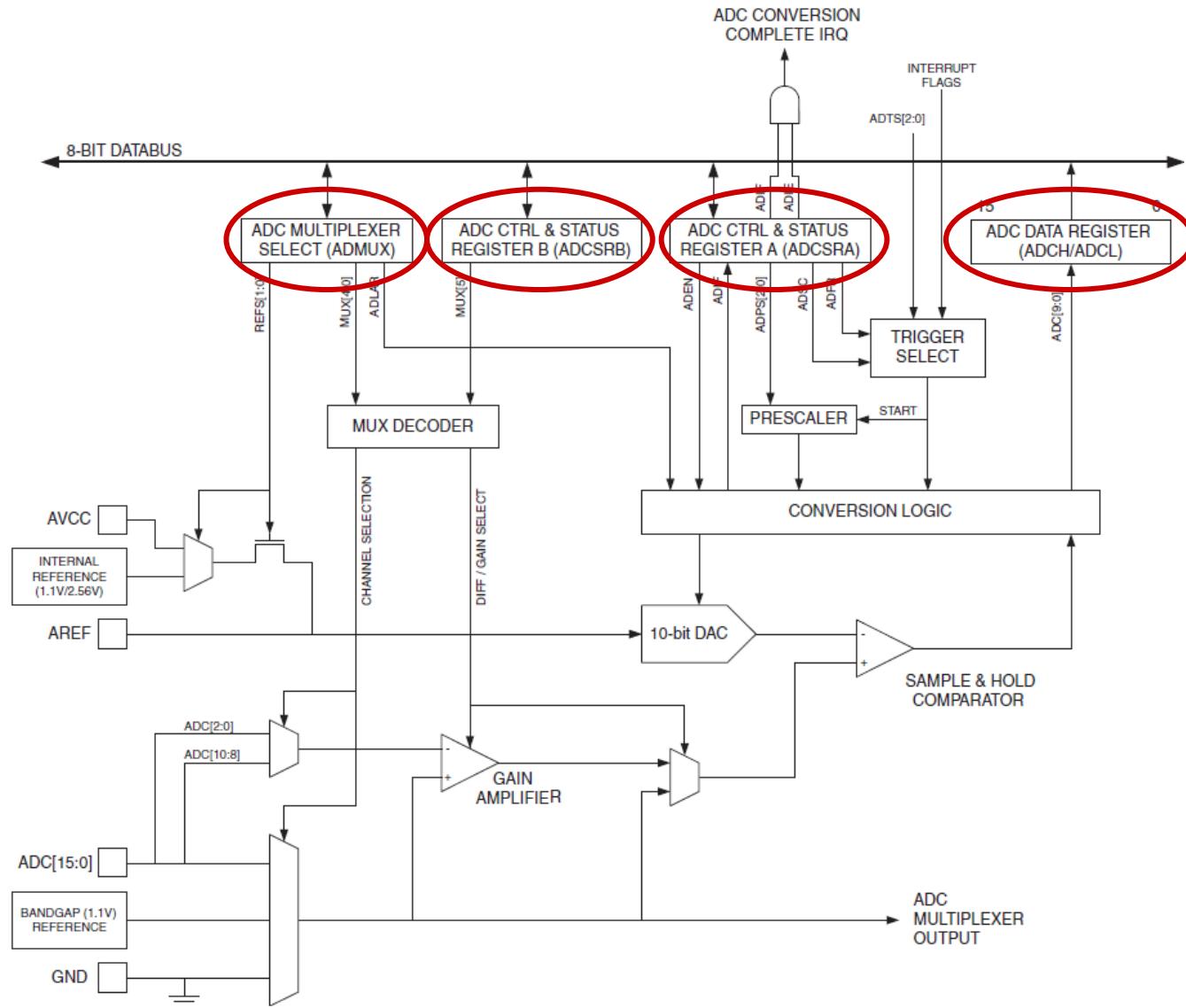
Successive approximation



Successive approximation ADC



(Blokdiagram, Mega2560 ADC)

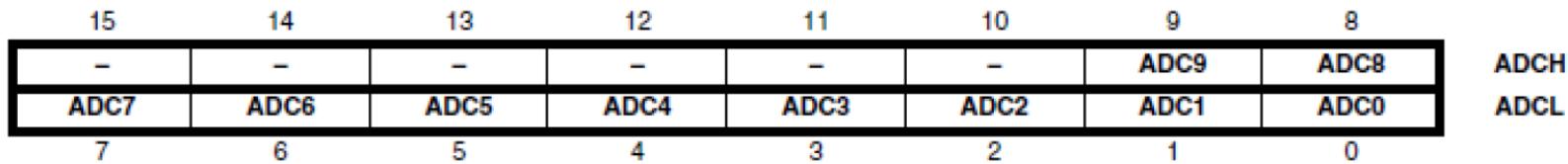


ADC enable

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

- Før ADC'en kan anvendes, skal den **enables** (skriv **1** til **ADCSRA bit 7**).
- Herved "tændes" for ADC'ens hardware.

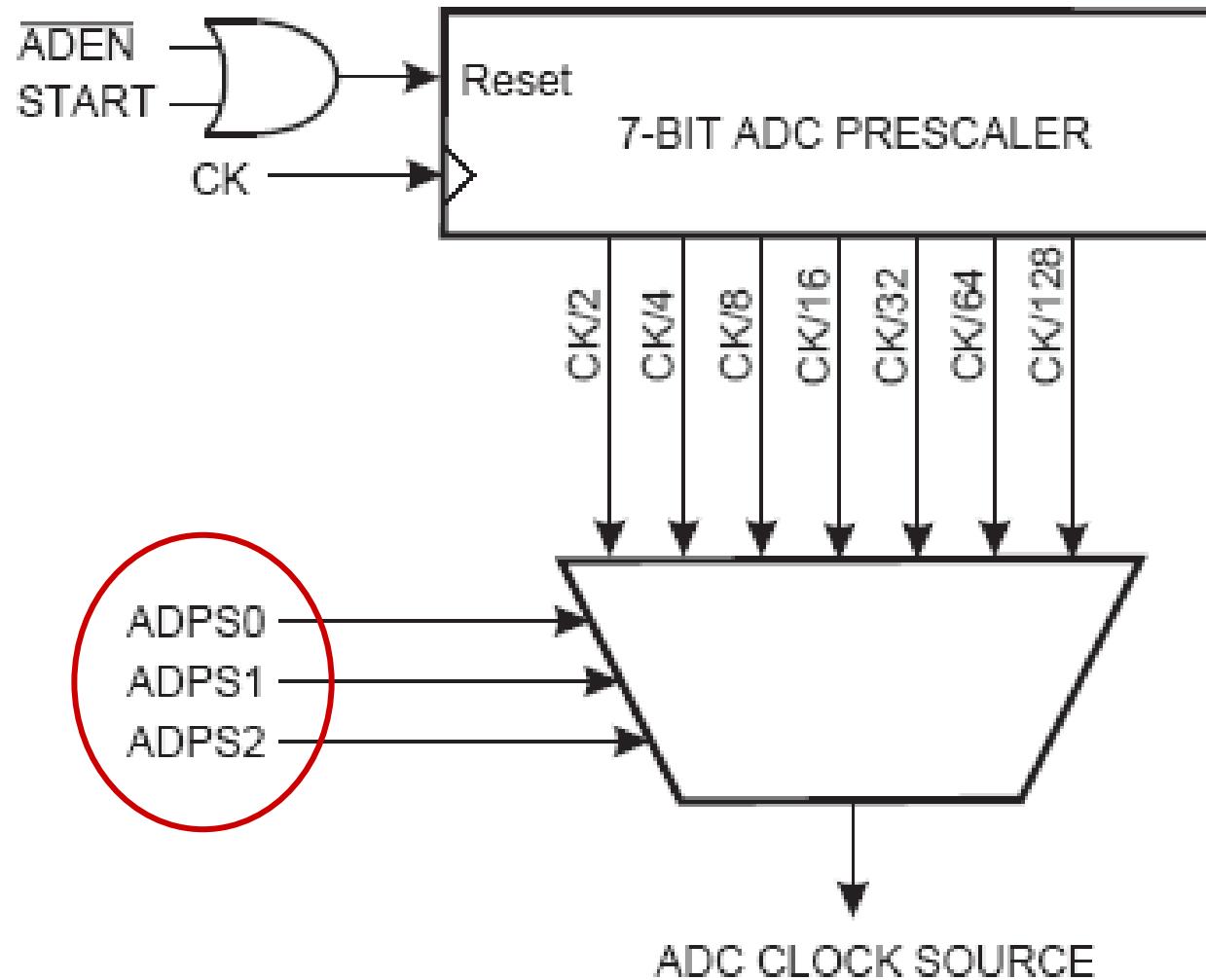
"ADCW" = ADCH → ADCL



- A/D konverterens resultat (efter konvertering) vil være i registrene **ADCH** og **ADCL**.
- Kan kun aflæses (ikke skrives til).
- Vores compiler kan aflæse til en variabel på denne måde:

```
unsigned int x;  
x = ADCW;
```

ADC prescaler / clock



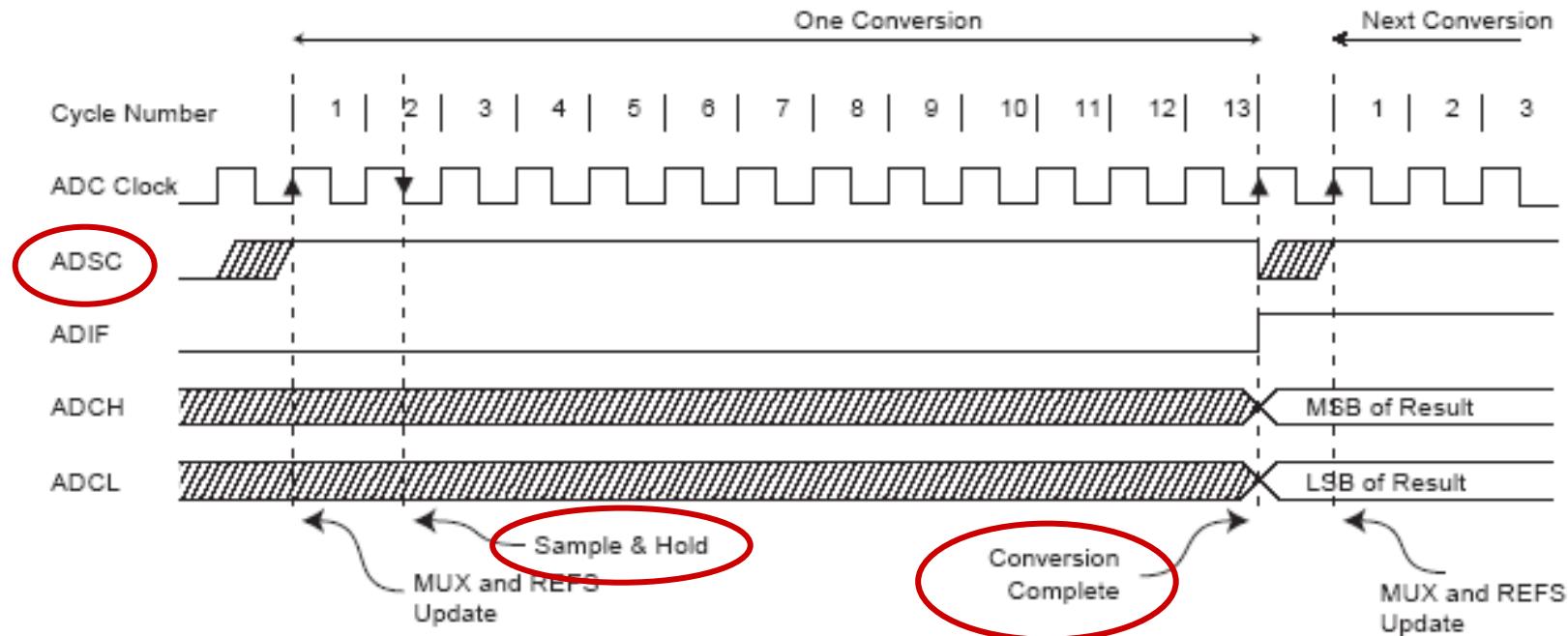
ADC prescaler

Bit	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	ADCSRA
Initial Value	0	0	0	0	0	0	0	0	
ADPS2			ADPS1			ADPS0			Division Factor
0	0	0	0	0	0	0	0	0	2
0	0	0	0	1	0	0	0	0	2
0	0	1	0	0	0	0	0	0	4
0	0	1	1	0	0	0	0	0	8
1	0	0	0	0	0	0	0	0	16
1	0	0	1	0	0	0	0	0	32
1	0	1	0	0	0	0	0	0	64
1	0	1	1	0	0	0	0	0	128

OBS : ADC clock skal være mellem
50 kHz og 200 kHz !

(For at kunne opnå 10 bit præcision).

ADC timing



Konverterings-tid

Condition	Sample & Hold (Cycles from Start of Conversion)	Conversion Time (Cycles)
First conversion	14.5	25
Normal conversions, single ended	1.5	13
Auto Triggered conversions	2	13.5
Normal conversions, differential	1.5/2.5	13/14

Eksempel :

$$f_{ADC} = 200 \text{ kHz} \Rightarrow \\ 13 \text{ cykles} = 65 \mu\text{s} \Rightarrow$$

15380 konverteringer / sekund

Test ("socrative.com"): Room = AUQUIZ

- CPU clock frekvens = 16 MHz.
ADC prescaler = 128.
Antag 13 ADC clockperioder per konvertering.
Hvor lang tid varer en A/D-konvertering ?

A:

8 mikrosekunder

B:

104 mikrosekunder

C:

81 nanosekunder

D:

16 mikrosekunder



Mega2560: Valg af reference, V_{ref}

Bit	7	6	5	4	3	2	1	0	ADMUX
Read/Write	R/W								
Initial Value	0	0	0	0	0	0	0	0	

REFS1	REFS0	Voltage Reference Selection ⁽¹⁾	
0	0	AREF, Internal V_{REF} turned off	AREF = Ekstern pin
0	1	AVCC with external capacitor at AREF pin	AVCC = Forsyning!
1	0	Internal 1.1V Voltage Reference with external capacitor at AREF pin	Intern 1,1 volt
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin	Intern 2,56 volt

Test ("socrative.com"): Room = AUQUIZ

- Vi bruger ADC'en til at måle en fast DC spænding. ADC'ens output er 400, når spændingen er 2 volt. Hvad er ADC'ens output, når spændingen ændres til 1 volt ?

A:

800

B:

400

C:

200

D:

100



Test ("socrative.com"): Room = AUQUIZ

- Vi bruger ADC'en til at måle en fast DC spænding.
ADC'en output er 400, når referencen er 4 volt.
Hvad er ADC'ens output, når referencen ændres til 2 volt ?

A:

200

B:

300

C:

400

D:

800



Mega2560: ADC pins (= PF og PK ben)



Mega2560: Valg af ADC-indgang (1)

7	6	5	4	3	2	1	0	
REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	
7	6	5	4	3	2	1	0	
-	ACME	-	-	MUX5	ADTS2	ADTS1	ADTS0	ADCSCR
R	R/W	R	R	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	

MUX5:0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
000000	ADC0			
000001	ADC1			
000010	ADC2			
000011	ADC3			
000100	ADC4			
000101	ADC5			
000110	ADC6			
000111	ADC7			

"Single ended" = En spænding refereret til stel.

Mega2560: Valg af ADC-indgange (2)

MUX5:0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
001000 ⁽¹⁾	N/A	ADC0	ADC0	10x
001001 ⁽¹⁾		ADC1	ADC0	10x
001010 ⁽¹⁾		ADC0	ADC0	200x
001011 ⁽¹⁾		ADC1	ADC0	200x
001100 ⁽¹⁾		ADC2	ADC2	10x
001101 ⁽¹⁾		ADC3	ADC2	10x
001110 ⁽¹⁾		ADC2	ADC2	200x
001111 ⁽¹⁾		ADC3	ADC2	200x
010000		ADC0	ADC1	1x
010001		ADC1	ADC1	1x
010010		ADC2	ADC1	1x
010011		ADC3	ADC1	1x
010100		ADC4	ADC1	1x
010101		ADC5	ADC1	1x
010110		ADC6	ADC1	1x
010111		ADC7	ADC1	1x
011000	N/A	ADC0	ADC2	1x
011001		ADC1	ADC2	1x
011010		ADC2	ADC2	1x
011011		ADC3	ADC2	1x
011100		ADC4	ADC2	1x
011101		ADC5	ADC2	1x

Mega2560: Valg af ADC-indgange (3)

MUX5:0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
011110	1.1V (V_{BG})			
011111	0V (GND)		N/A	
100000	ADC8			
100001	ADC9			
100010	ADC10			
100011	ADC11		N/A	
100100	ADC12			
100101	ADC13			
100110	ADC14			
100111	ADC15			

Mega2560: Valg af ADC-indgange (4)

MUX5:0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
101000 ⁽¹⁾		ADC8	ADC8	10x
101001 ⁽¹⁾		ADC9	ADC8	10x
101010 ⁽¹⁾		ADC8	ADC8	200x
101011 ⁽¹⁾		ADC9	ADC8	200x
101100 ⁽¹⁾		ADC10	ADC10	10x
101101 ⁽¹⁾		ADC11	ADC10	10x
101110 ⁽¹⁾		ADC10	ADC10	200x
101111 ⁽¹⁾		ADC11	ADC10	200x
110000		ADC8	ADC9	1x
110001		ADC9	ADC9	1x
110010		ADC10	ADC9	1x
110011		ADC11	ADC9	1x
110100		ADC12	ADC9	1x
110101		ADC13	ADC9	1x
110110		ADC14	ADC9	1x
110111		ADC15	ADC9	1x
111000		ADC8	ADC10	1x
111001		ADC9	ADC10	1x
111010		ADC10	ADC10	1x
111011		ADC11	ADC10	1x
111100		ADC12	ADC10	1x
111101	N/A	ADC13	ADC10	1x
111110	Reserved		N/A	
111111	Reserved		N/A	

Normal / "Left Adjust Result"

Bit	7	6	5	4	3	2	1	0	ADMUX
Read/Write	R/W								
Initial Value	0	0	0	0	0	0	0	0	

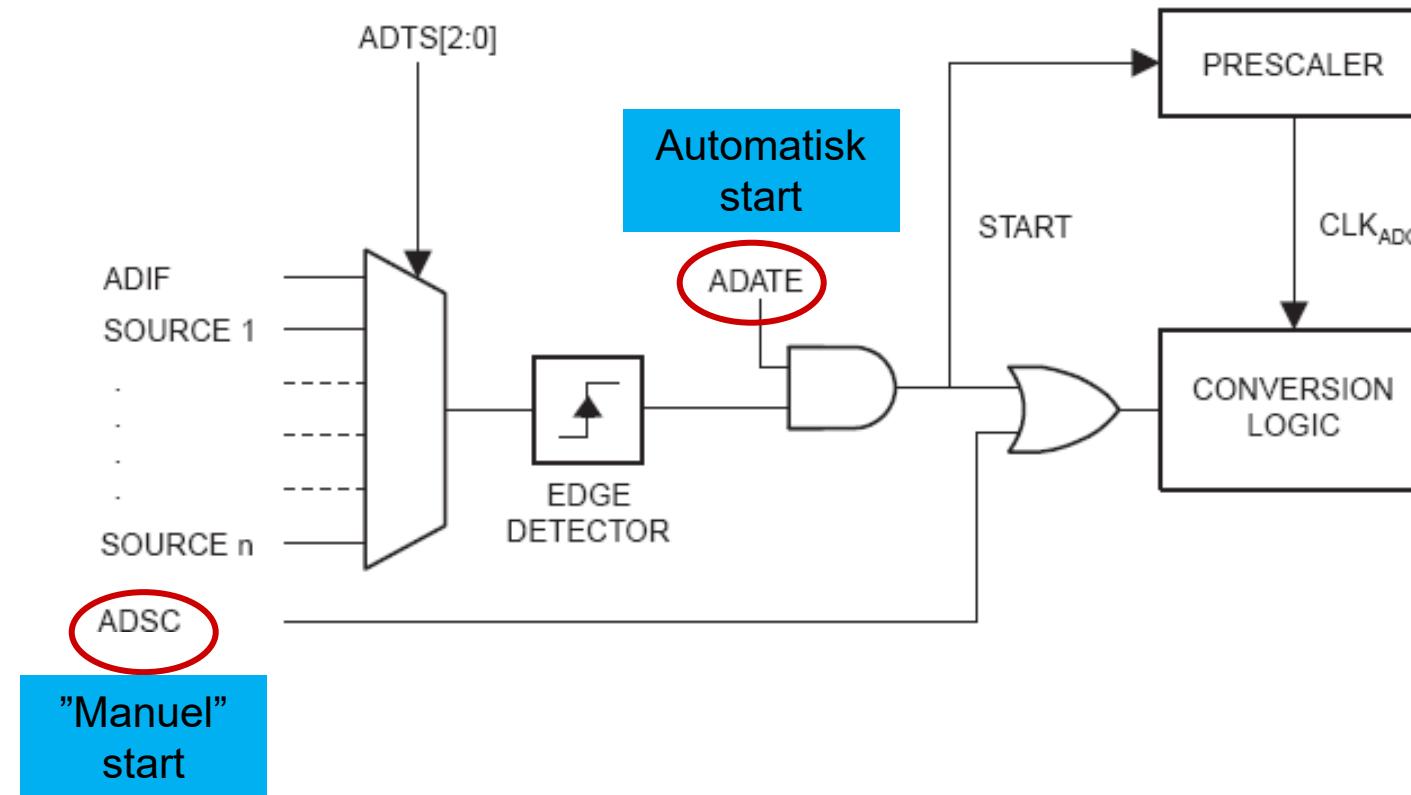
ADLAR = 0

Bit	15	14	13	12	11	10	9	8	ADCH
Read/Write	R	R	R	R	R	R	R	R	ADCL
Initial Value	0	0	0	0	0	0	0	0	

ADLAR = 1

Bit	15	14	13	12	11	10	9	8	ADCH
Read/Write	R	R	R	R	R	R	R	R	ADCL
Initial Value	0	0	0	0	0	0	0	0	

ADC start ("trigger")



"Manuel" START

Bit	7	6	5	4	3	2	1	0	
ReadWrite	R/W	ADCSRA							
Initial Value	0	0	0	0	0	0	0	0	

- Bit ADSC = 1 : Konvertering starter.
ADCSRA |= 0b01000000;

- Bit ADSC == 0 : Konvertering slut.
**while (ADCSRA & 0b01000000)
{ }
// Herefter kan ADC aflæses
x = ADCW;**

Mega2560: Automatisk START ("trigger")



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

7	6	5	4	3	2	1	0	
-	ACME	-	-	MUX5	ADTS2	ADTS1	ADTS0	ADCSR _B
R	R/W	R	R	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	

ADTS2	ADTS1	ADTS0	Trigger Source ← "Interrupt fra..."
0	0	0	Free Running mode
0	0	1	Analog Comparator
0	1	0	External Interrupt Request 0
0	1	1	Timer/Counter0 Compare Match A
1	0	0	Timer/Counter0 Overflow
1	0	1	Timer/Counter1 Compare Match B
1	1	0	Timer/Counter1 Overflow
1	1	1	Timer/Counter1 Capture Event

ADC interrupt

Bit	7	6	5	4	3	2	1	0	
Read/Write	R/W	ADCSRA							
Initial Value	0	0	0	0	0	0	0	0	

- **ADIE**: "ADC interrupt enable".
1 => Der genereres ADC-interrupt, hvis global interrupt enable også er sat.
- (ADIF: "ADC interrupt flag")
Sættes høj efter hver konvertering.
Nulstilles automatisk i interruptrutinen ELLER
ved at skrive et 1-tal til bit ADIF.

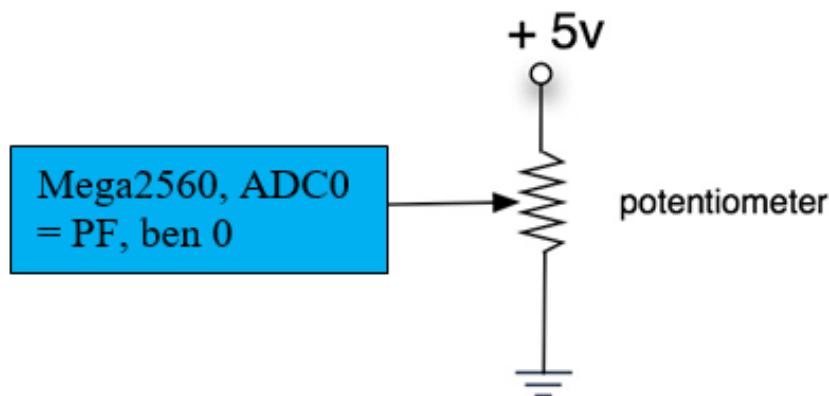
LAB13



Der skal anvendes en skruetrækker med en speciel lille kærv for at justere på potentiometeret.

OBS: Brug ikke for mange kræfter!
Du kan komme til at ødelægge
potentiometeret.

Skruetrækkeren vil kunne lånes til øvelsen.



Ved at skrue på potentiometeret kan vi justere spændingen på ADC0 benet mellem 0 og 5 volt.

LAB13, del 1

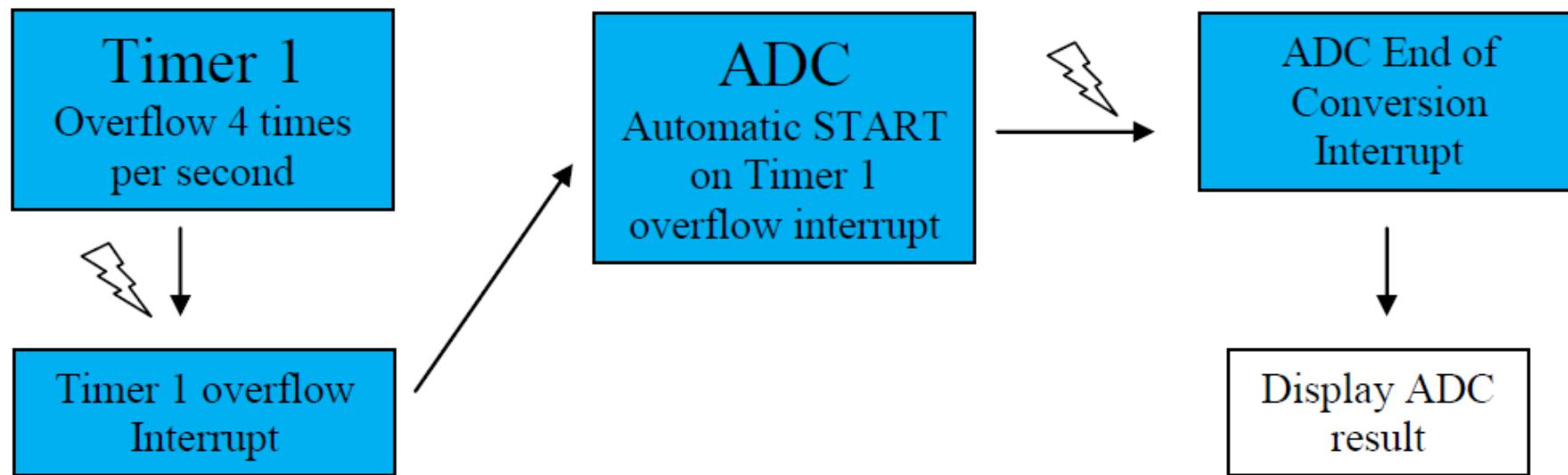


A screenshot of a terminal window showing a series of ADC0 readings. The window has a dark header bar and a light gray body. The text is in a monospaced font.

```
ADC0 = 799  
ADC0 = 780  
ADC0 = 765  
ADC0 = 739  
ADC0 = 726  
ADC0 = 710
```

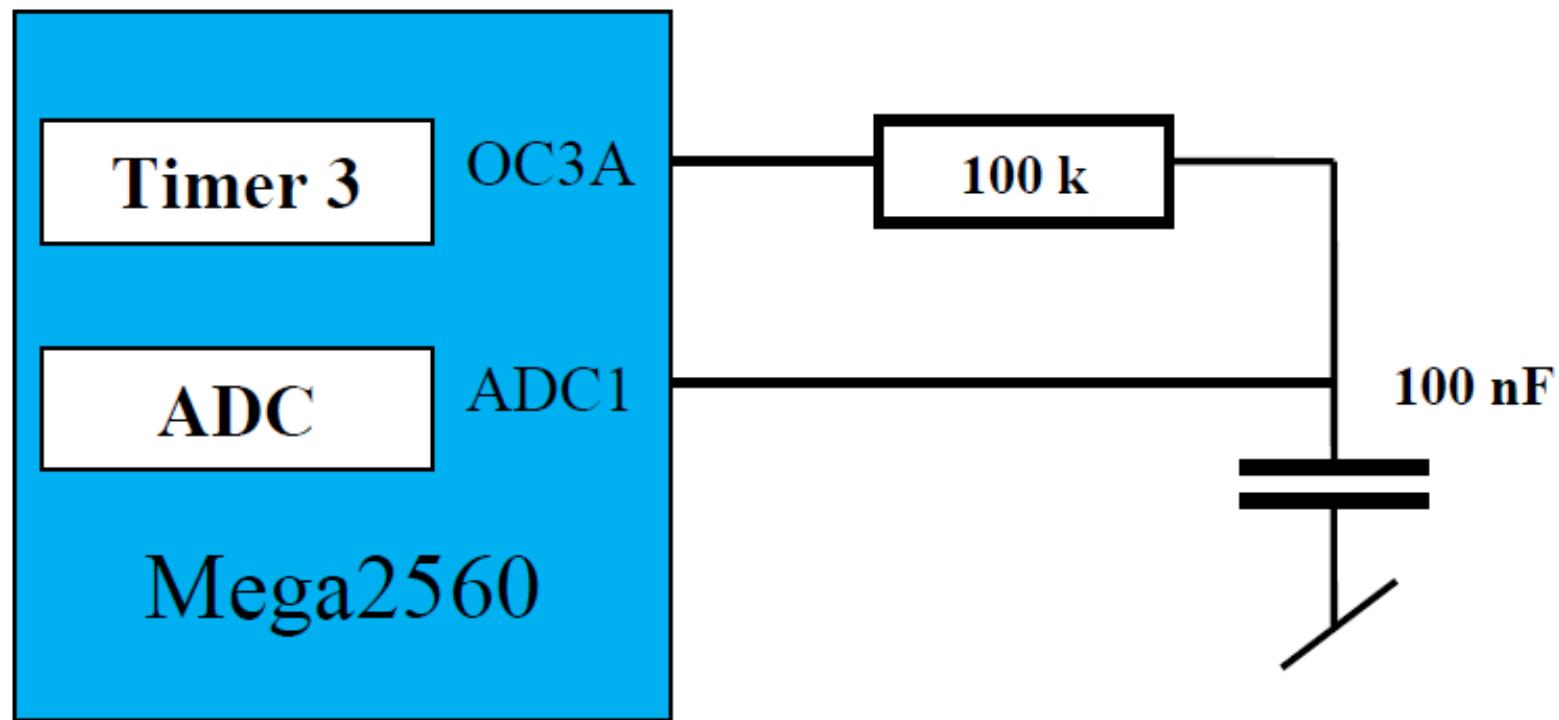
- "Manuel" start
- Aflæs og display potentiometer-spændingen

LAB13, del 2

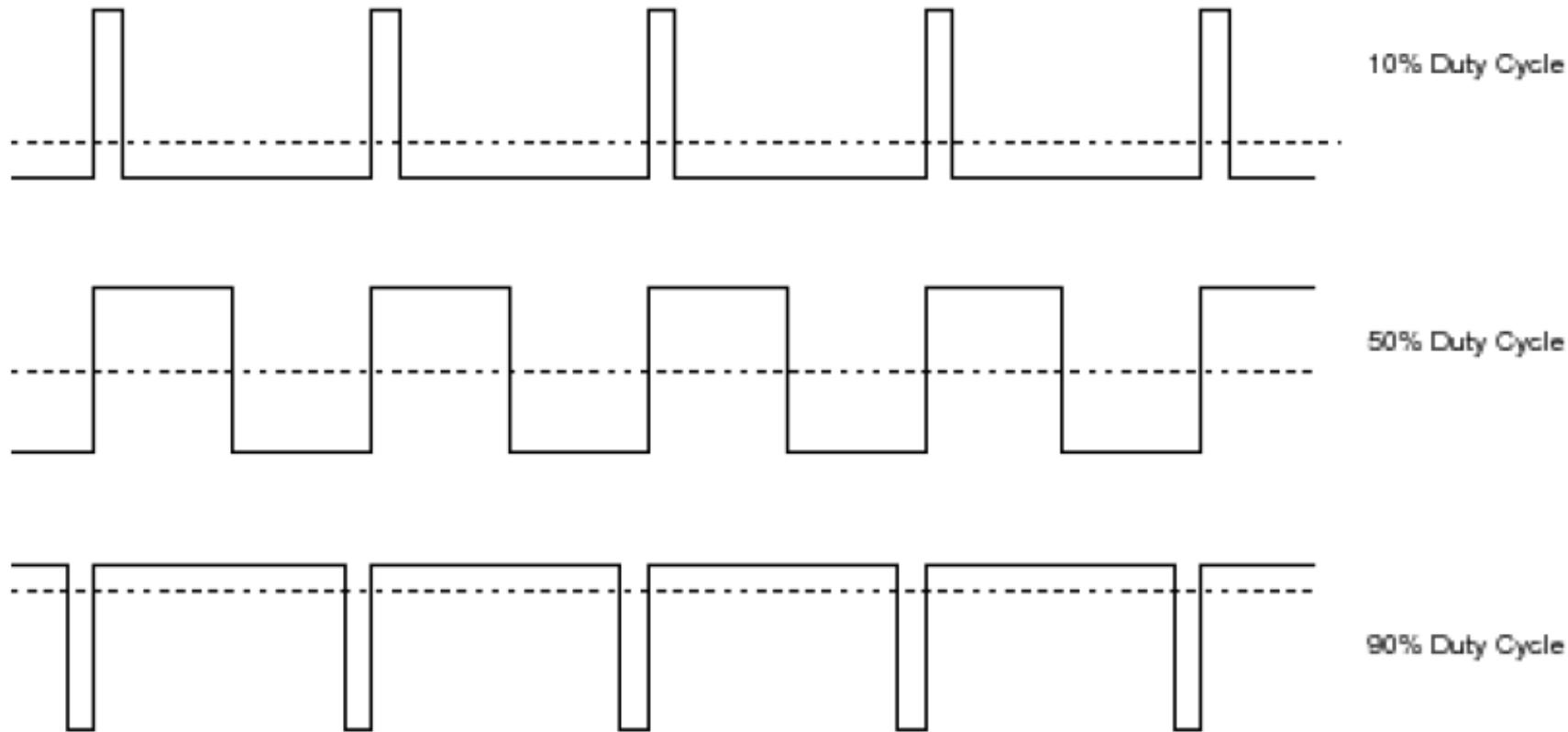


- Automatisk "Start on Timer 1 overflow"
- ADC interrupt enabled
- Aflæs og display potentiometer-spændingen

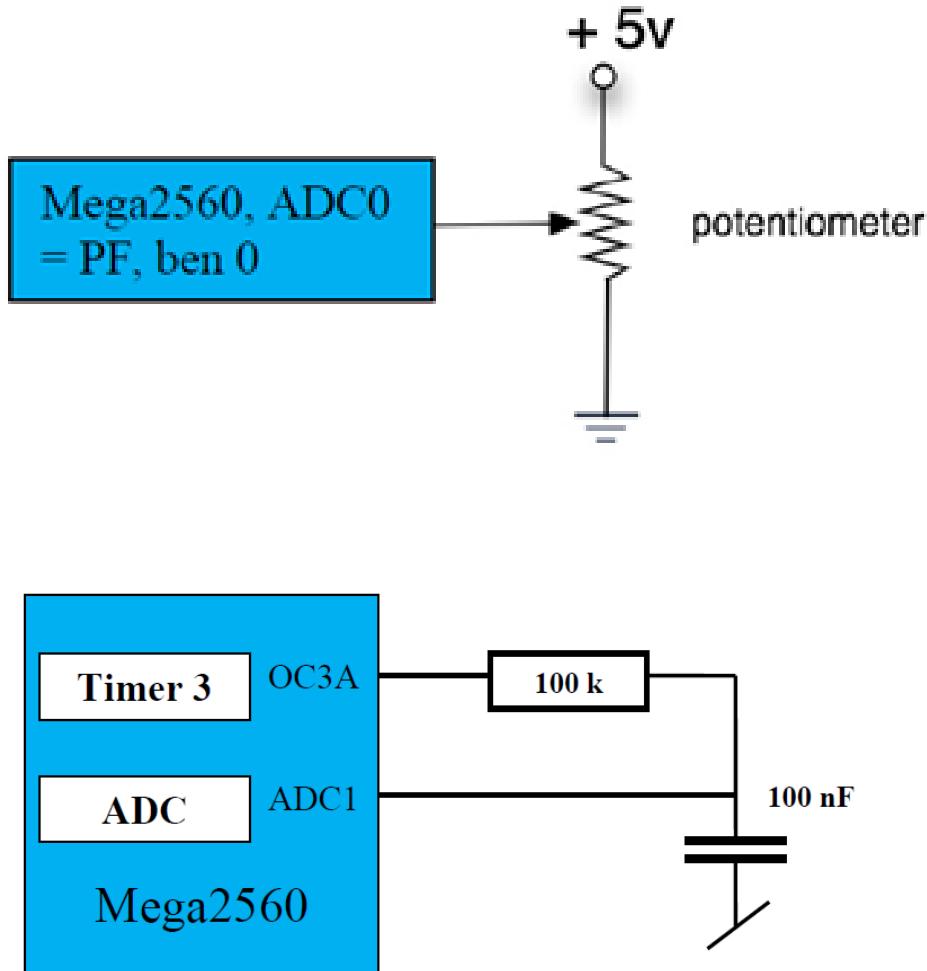
LAB13, del 3



LAB13, del 3



LAB13, del 3



Simpel regulering :

Juster spændingen over kondensatoren, så den bliver den samme som spændingen fra potentiometeret.

LAB13, del 3

```
ADC0: 192 424 :ADC1  
ADC0: 192 425 :ADC1  
ADC0: 192 425 :ADC1  
ADC0: 192 424 :ADC1  
ADC0: 192 422 :ADC1  
ADC0: 192 421 :ADC1  
ADC0: 192 419 :ADC1  
ADC0: 192 417 :ADC1  
ADC0: 192 417 :ADC1
```

- ADC0 er måling fra potentiometeret
- ADC1 er måling fra kondensatoren

MCP9700



**MCP9700/9700A
MCP9701/9701A**

Low-Power Linear Active Thermistor ICs

General Description

MCP9700/9700A and MCP9701/9701A sensors with Linear Active Thermistor Integrated Circuit (IC) comprise a family of analog temperature sensors that convert temperature to analog voltage.

MCP9700

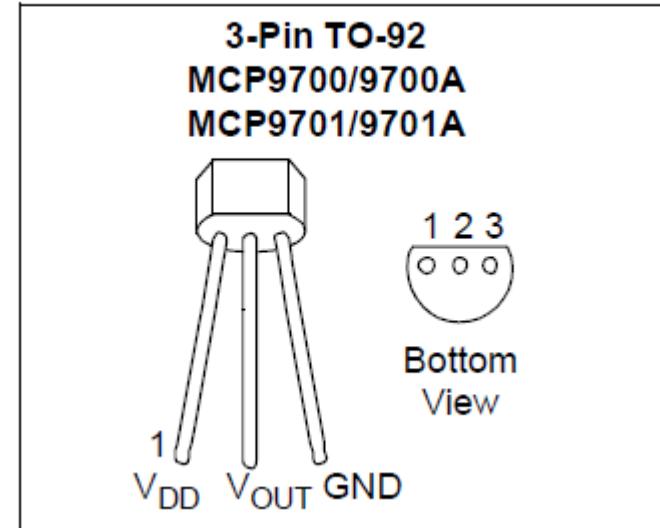
Features

- Tiny Analog Temperature Sensor
- Available Packages:
 - SC70-5, SOT-23-3, TO-92-3
- Wide Temperature Measurement Range:
 - -40°C to +125°C (Extended Temperature)
 - -40°C to +150°C (High Temperature)
(MCP9700, SOT-23-3 and SC70-5 only)
- Accuracy:
 - $\pm 2^\circ\text{C}$ (max.), 0°C to +70°C (**MCP9700A/9701A**)
 - $\pm 4^\circ\text{C}$ (max.), 0°C to +70°C (**MCP9700/9701**)
- Optimized for Analog-to-Digital Converters (ADCs):
 - 10.0 mV/°C (typical) (**MCP9700/9700A**)
 - 19.5 mV/°C (typical) (**MCP9701/9701A**)
- Wide Operating Voltage Range:
 - $V_{DD} = 2.3\text{V}$ to 5.5V (**MCP9700/9700A**)
 - $V_{DD} = 3.1\text{V}$ to 5.5V (**MCP9701/9701A**)
- Low Operating Current: 6 μA (typical)
- Optimized to Drive Large Capacitive Loads

Typical Applications

- Hard Disk Drives and Other PC Peripherals
- Entertainment Systems
- Home Appliance
- Office Equipment
- Battery Packs and Portable Equipment
- General Purpose Temperature Monitoring

Package Types



MCP9700

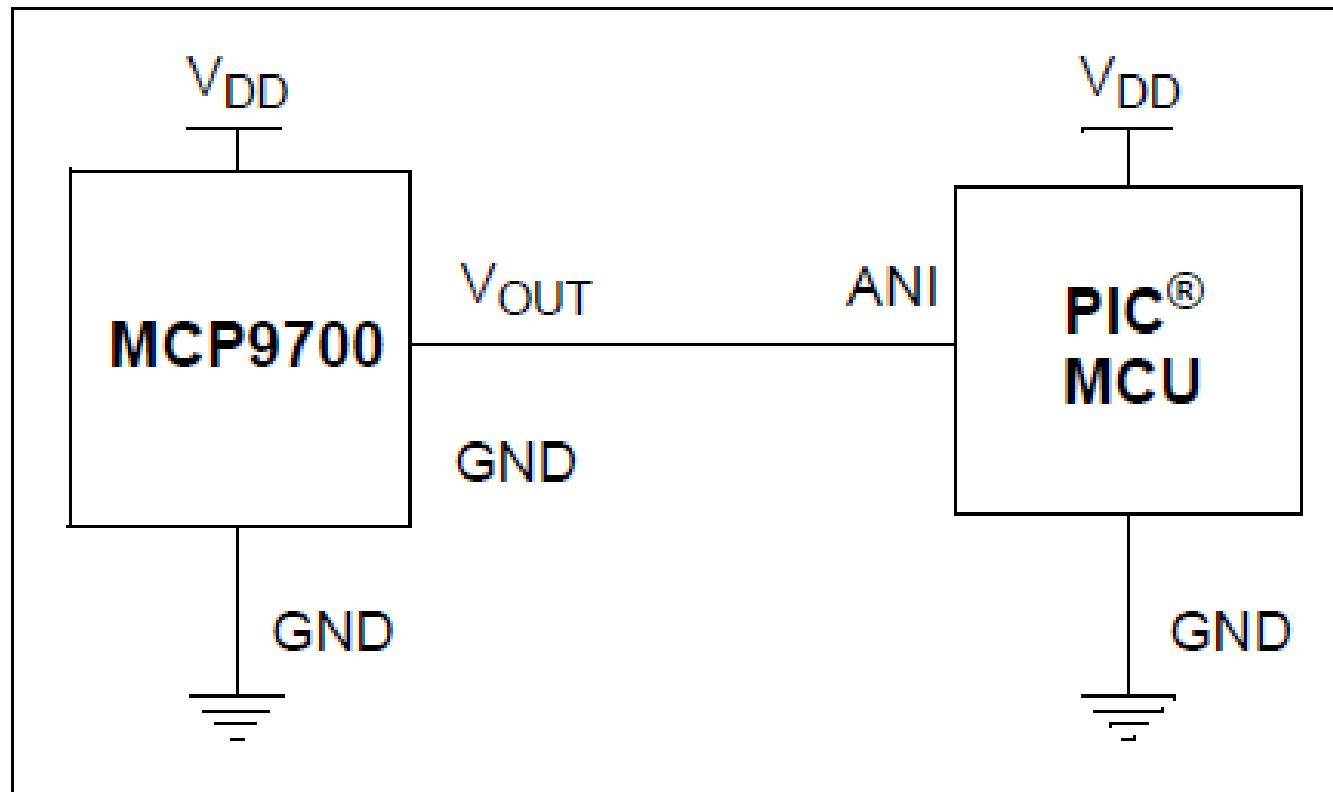


FIGURE 4-1: Typical Application Circuit.

Do not use ADC0 and ADC1!

MOSFET as a switch

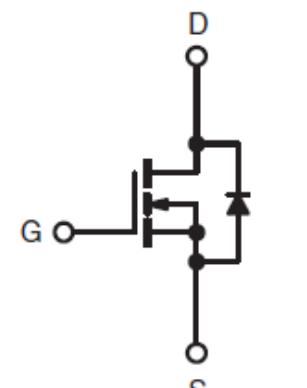
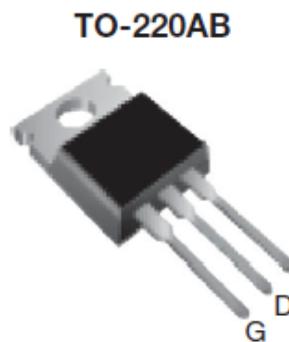


IRLZ44, SiHLZ44

Vishay Siliconix

Power MOSFET

PRODUCT SUMMARY		
V_{DS} (V)		60
$R_{DS(on)}$ (Ω)	$V_{GS} = 5.0$ V	0.028



N-Channel MOSFET

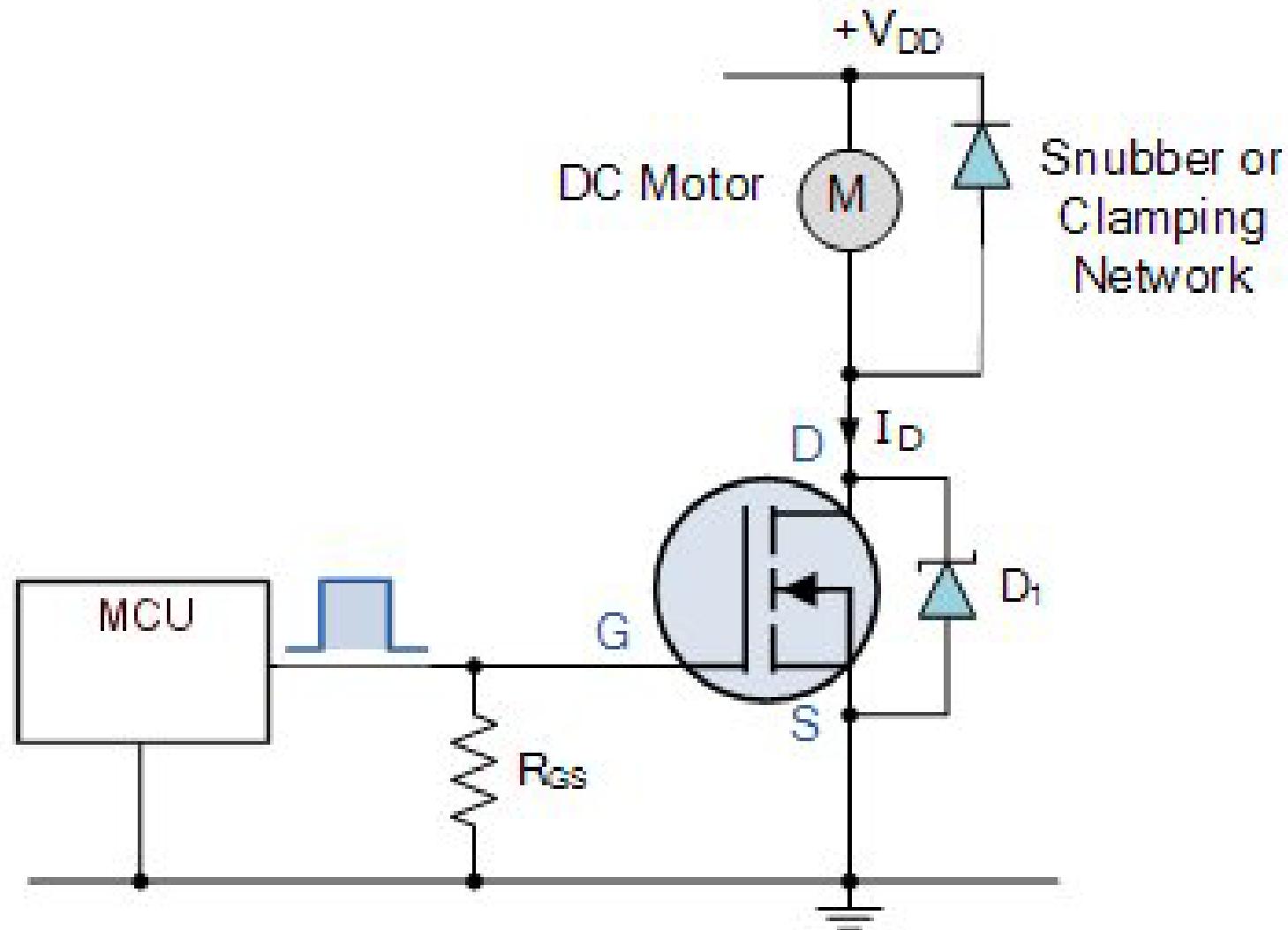
FEATURES

- Dynamic dV/dt Rating
- Logic-Level Gate Drive
- $R_{DS(on)}$ Specified at $V_{GS} = 4$ V and 5 V
- 175 °C Operating Temperature
- Fast Switching
- Ease of Paralleling
- Simple Drive Requirements
- Compliant to RoHS Directive 2002/95/EC

MOSFET as a switch

ABSOLUTE MAXIMUM RATINGS ($T_C = 25^\circ\text{C}$, unless otherwise noted)				
PARAMETER		SYMBOL	LIMIT	UNIT
Drain-Source Voltage		V_{DS}	60	V
Gate-Source Voltage		V_{GS}	± 10	
Continuous Drain Current ^e	V_{GS} at 5.0 V	$T_C = 25^\circ\text{C}$	50	A
Continuous Drain Current		$T_C = 100^\circ\text{C}$	36	
Pulsed Drain Current ^a		I_{DM}	200	
Linear Derating Factor			1.0	W/ $^\circ\text{C}$
Single Pulse Avalanche Energy ^b		E_{AS}	400	mJ
Maximum Power Dissipation	$T_C = 25^\circ\text{C}$	P_D	150	W
Peak Diode Recovery dV/dt ^c		dV/dt	4.5	V/ns
Operating Junction and Storage Temperature Range		T_J, T_{stg}	- 55 to + 175	$^\circ\text{C}$
Soldering Recommendations (Peak Temperature) ^d	for 10 s		300	
Mounting Torque	6-32 or M3 screw		10	lbf · in
			1.1	N · m

MOSFET as a switch



Ref.: http://www.electronics-tutorials.ws/transistor/tran_7.html





AARHUS
UNIVERSITY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING