



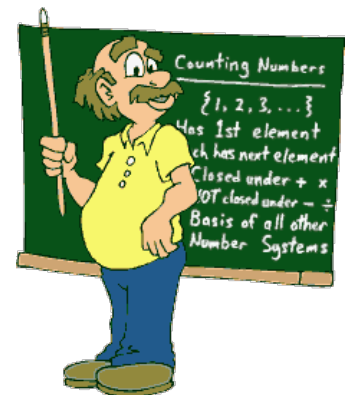
AARHUS
UNIVERSITY
SCHOOL OF ENGINEERING

MSYS

Microcontroller Systems

Lektion 10

C programming



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 (begrænset).

AVR GCC **er integreret i Atmel Studio** ☺

Andre C compilers for Atmel AVR (ikke gratis):

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

Data typer i C

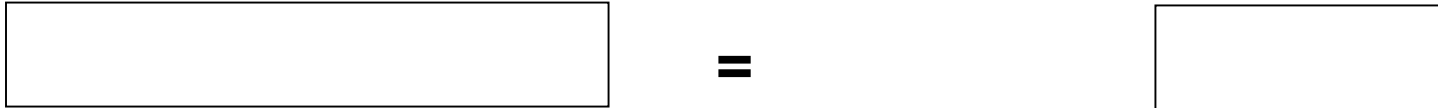
Data Type	Size in Bits	Data Range/Usage
unsigned char	8-bit	0 to 255
char	8-bit	-128 to +127
unsigned int	16-bit	0 to 65,535
int	16-bit	-32,768 to +32,767
unsigned long	32-bit	0 to 4,294,967,295
long	32-bit	-2,147,483,648 to +2,147,483,648
float	32-bit	$\pm 1.175e-38$ to $\pm 3.402e38$
double	32-bit	$\pm 1.175e-38$ to $\pm 3.402e38$

Brug altid **mindst mulige type** (helst unsigned char) !

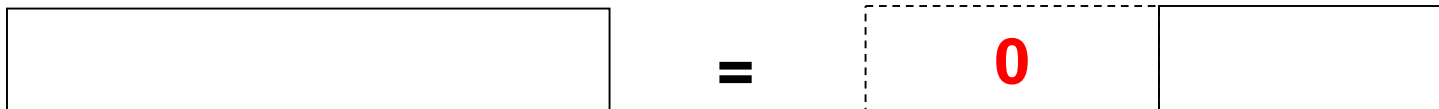
Hvorfor?

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:

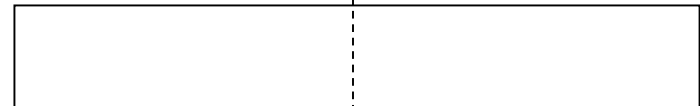


Fra stor til lille type

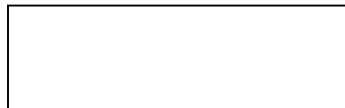
- Hvis en **variabel af en mindre type sættes lig med en variabel af større type**, mistes de mest betydende bits !
- Den "store variabel" opfattes altså midlertidigt som om den har den "lille variabels" type.



=



vil blive udført som:



=



OBS !

Ignoreres

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;
```



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

- 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

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 ansvar* at sørge for, at dette ikke sker !



Eksempel

```
unsigned long z;  
unsigned long x = 15000;  
unsigned int y = 63000;
```

```
// Hvorfor kan dette gå galt ?  
z = x + (10 * y);
```

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

Type casting

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

```
unsigned long z;  
unsigned long x = 15000;  
unsigned int y = 63000;  
  
// Nu behandler compileren mellemresultatet som 32 bit  
// - og beregningen bliver korrekt.  
// Men variabelen 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 af clock frekvensen ☹️
- Tidsforsinkelsen afhænger helt af compileren ☹️
- Optimeringsgraden **SKAL** sættes til level 00 ☹️

Delays på primitiv vis

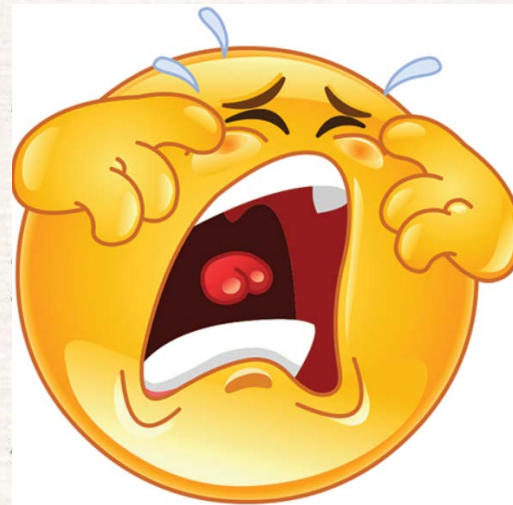
Example 7-7

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++);
}
```

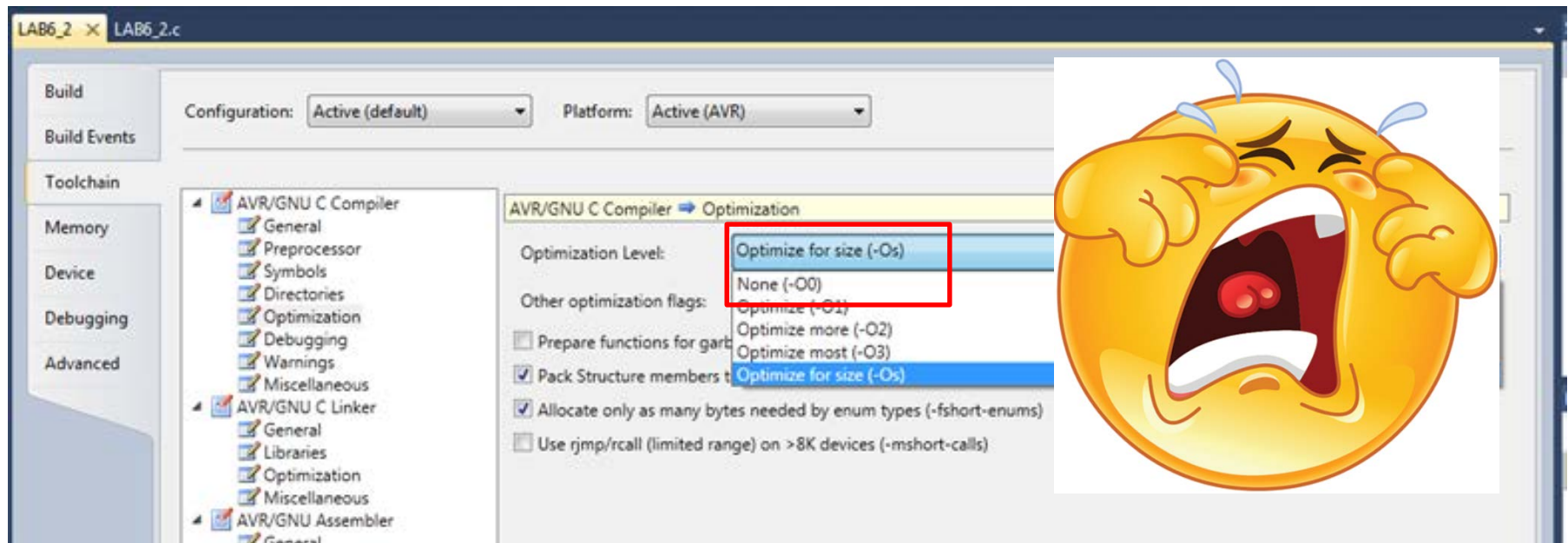
```
int main(void)
{
    DDRB = 0xFF;
    while (1)
    {
        PORTB = 0xAA;
        delay100ms();
        PORTB = 0x55;
        delay100ms();
    }
    return 0;
}
```



on your
e result.

**OBS: Compiler optimering
skal slås fra !
Hvorfor ?**

Sådan sættes optimering til "level O0"



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);
```

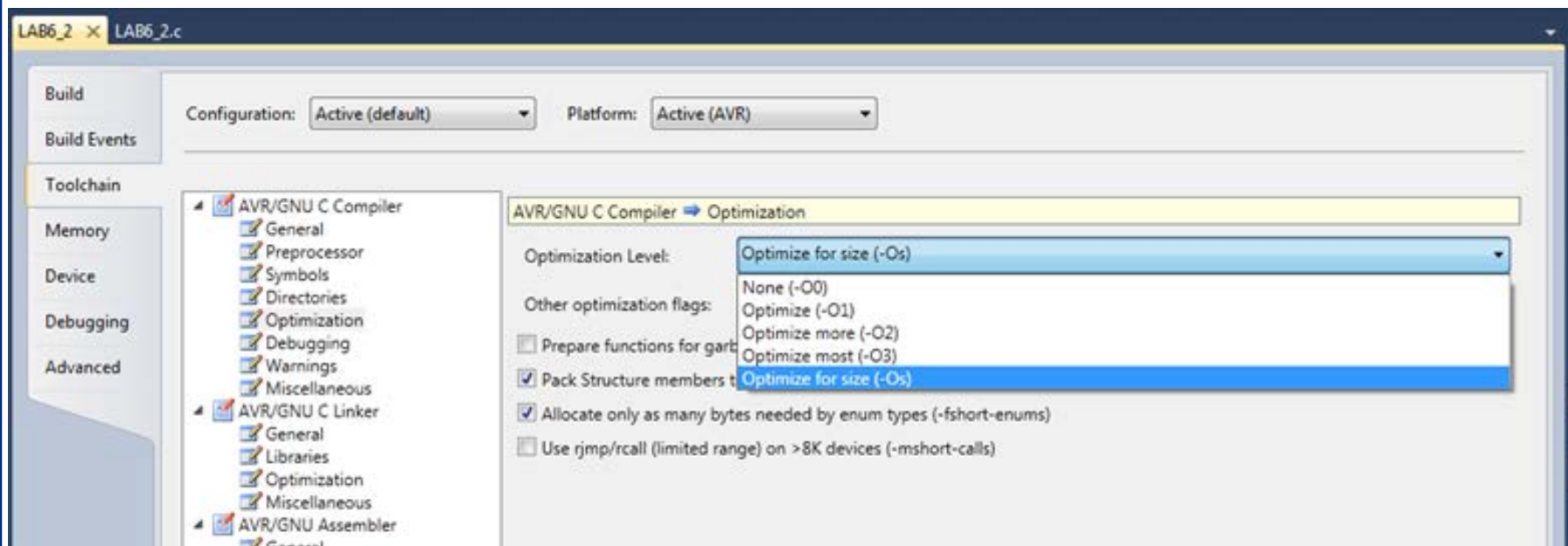


- Metoden er compiler afhængig – ikke hardware afhængig.

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= ~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

~ 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	<code>a &= b</code>	<code>a = a & b</code>
OR assignment	<code>a = b</code>	<code>a = a b</code>

```
#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) $1 \ll 7$
- (b) $1 \ll 2$
- (c) $1 \ll 4$
- (d) $\sim(1 \ll 5)$
- (e) $\sim(1 \ll 3)$
- (f) $\sim(1 \ll 1)$

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.

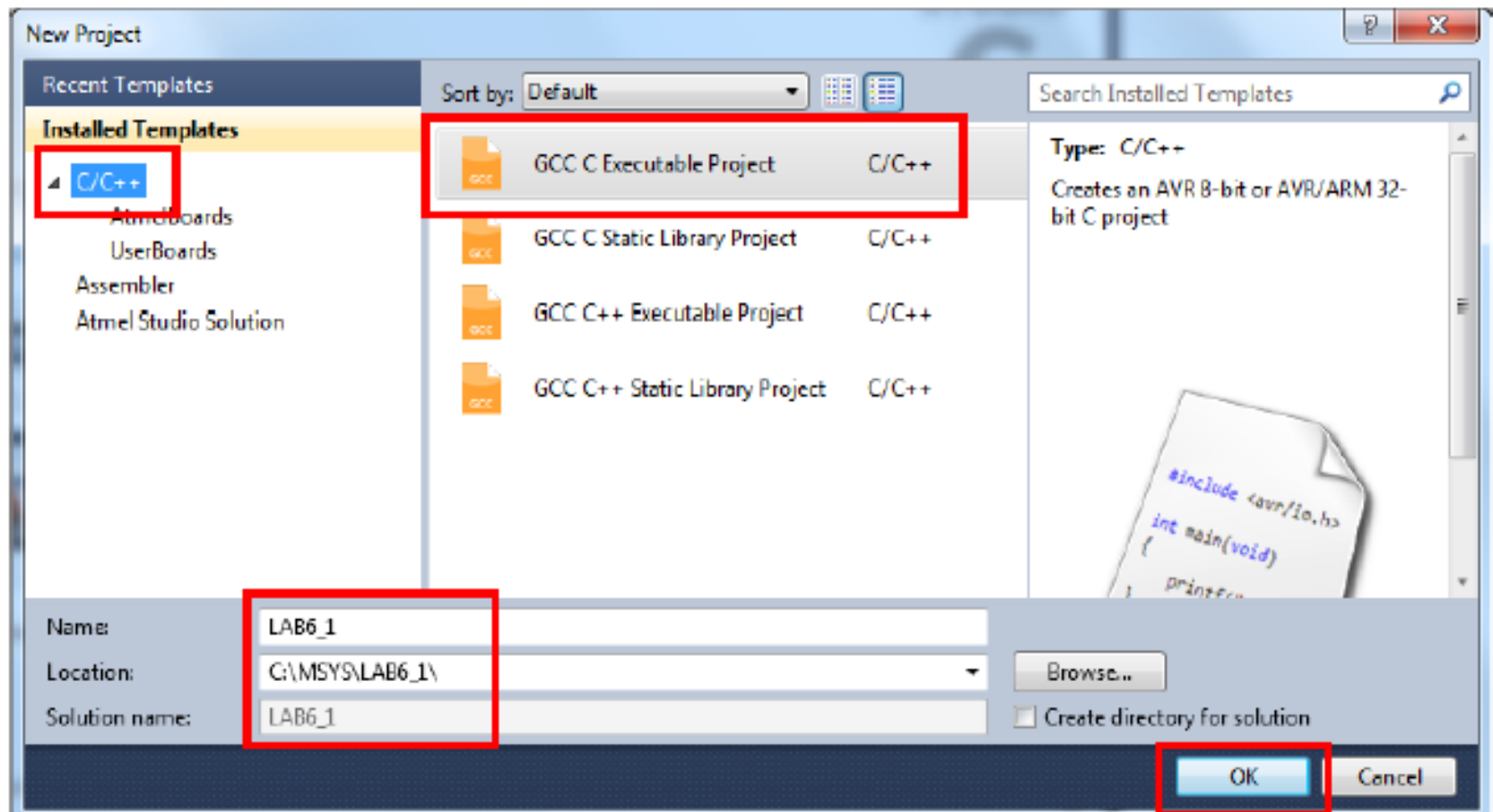
Solution:

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

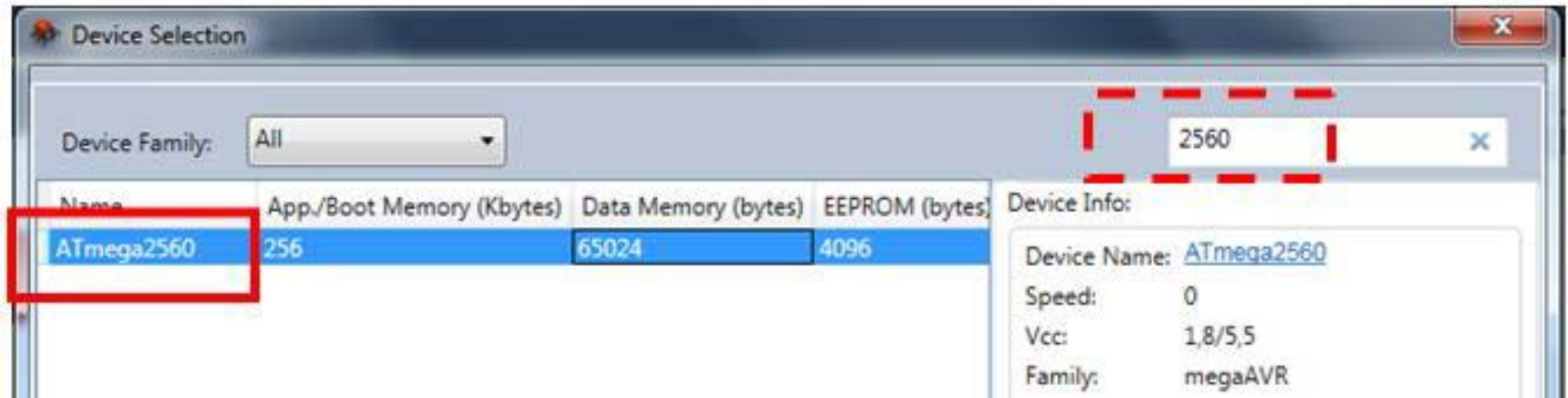
int main(void)
{
    DDRB = DDRB & ~(1<<SENSOR);    //SENSOR pin is input
    DDRC = DDRC | (1<< LED);        //LED pin is output

    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 Atmel Studio



ACR GCC C projekt i Atmel Studio

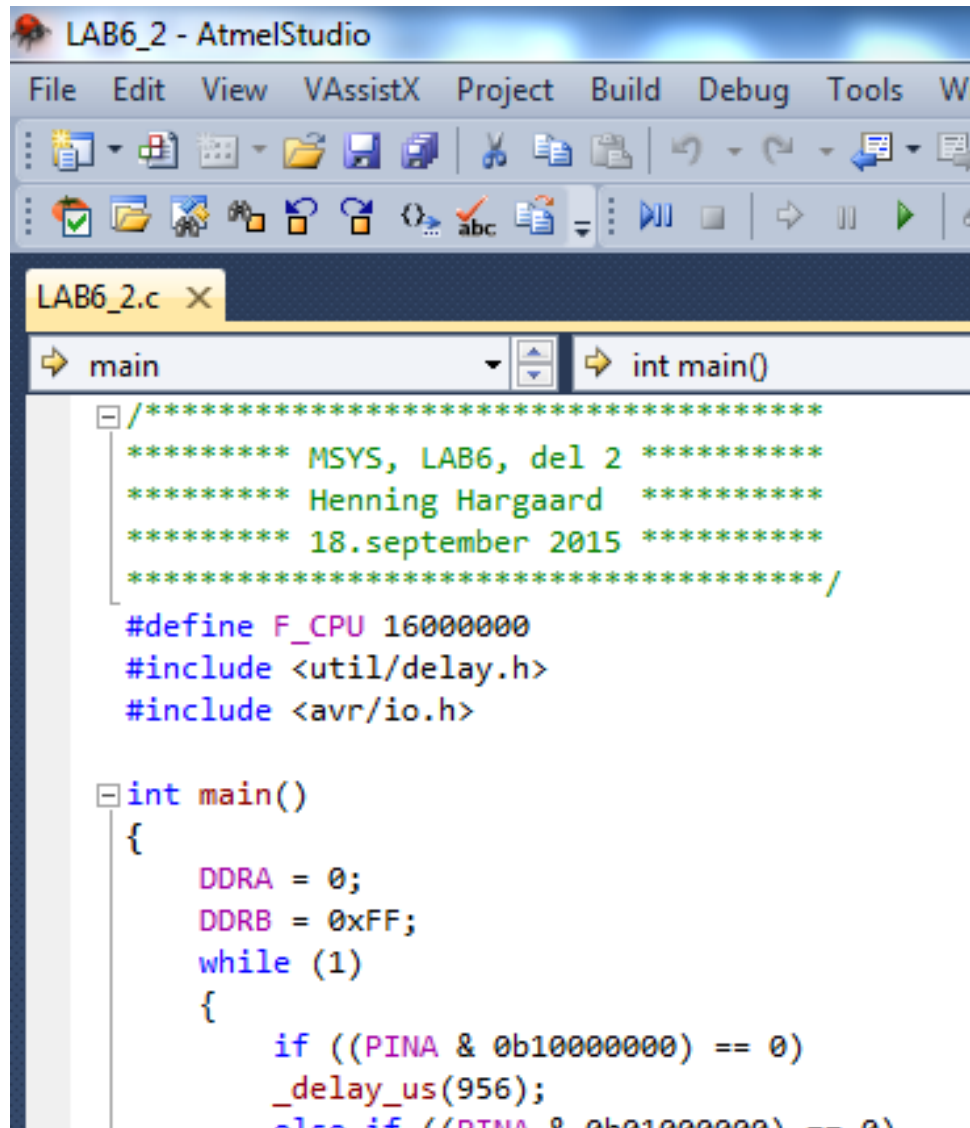


```
LAB6_1.c x
/*
 * LAB6_1.c
 *
 * Created: 18-09-2015 14:13:03
 * Author: hh
 */

#include <avr/io.h>

int main(void)
{
    while(1)
    {
        //TODO:: Please write your application code
    }
}
```

ACR GCC C projekt i Atmel Studio



The screenshot shows the Atmel Studio IDE with the file 'LAB6_2.c' open. The code is a C program for an AVR microcontroller, featuring a main function that initializes DDRA and DDRB, and enters a while loop with conditional delays.

```
LAB6_2 - AtmelStudio
File Edit View VAssistX Project Build Debug Tools Wi
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 ((PINA & 0b10000000) == 0)
            _delay_us(956);
        else if ((PINA & 0b01000000) == 0)
```

Slut på lektion 10

