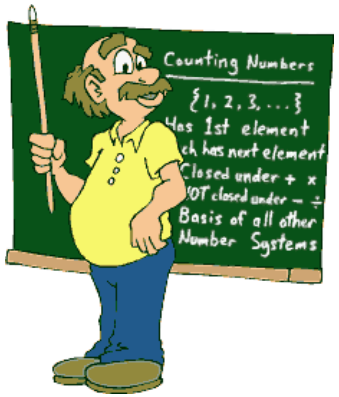
iha.dk

IECA

Embedded Computer Architecture

Lesson 10

C programming



Version: 6-11-2015, Henning Hargaard

GCC open source compilers

GCC = Gnu Compiler Collection
Open Source (UNIX/Linux):
☺ Free.
☹ Sometimes poor documentations.

AVR GCC: GCC Compiler for Atmel AVR. Most common is the C compiler, but C++ compiler also available.

AVR GCC is integrated in Atmel Studio ☺

Other compilers for Atmel AVR (not free):
- IAR (swedish). Very prof. Expensive.
- CodeVision (romanian). Less expensive.

Data types in 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	±1.175e-38 to ±3.402e38
double	32-bit	±1.175e-38 to ±3.402e38

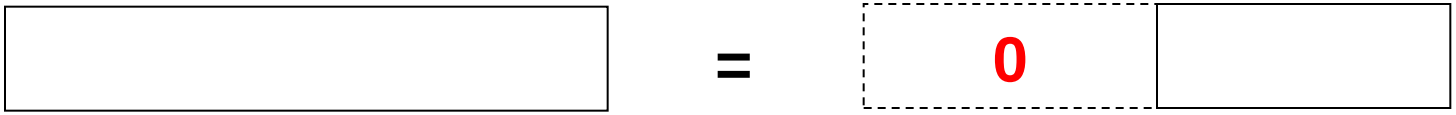
Always use the smallest possible type
(preferably unsigned char) !

Otherwise the code will be extensive and slow executing.

C rule: From smaller to larger type

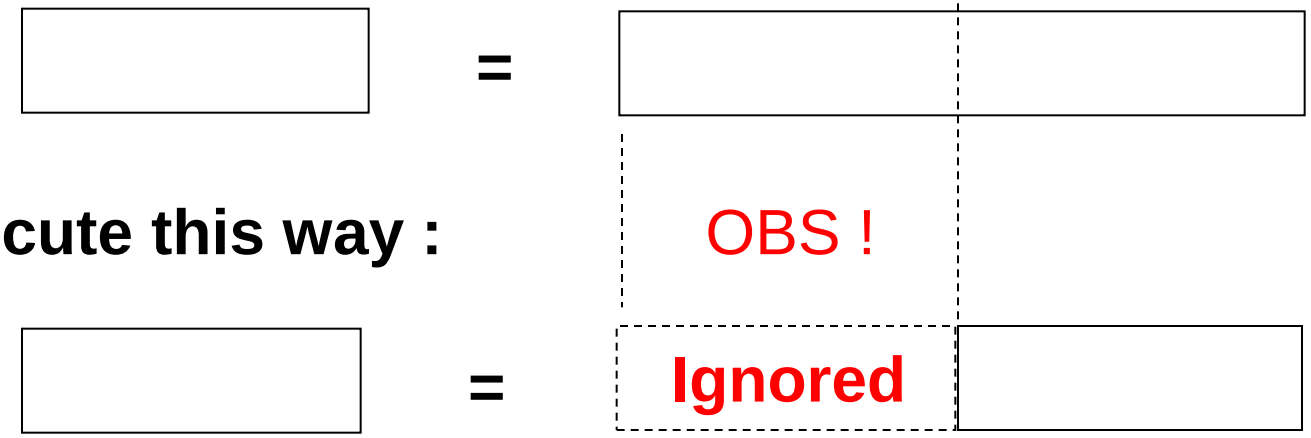
- When assigning a variable of smaller type to a variable of larger type, zeroes automatically will be appended ”in front”.
- The ”smaller variable” is temporarily regarded as having the type of the ”larger variable”.

will execute this way:



C rule: From larger to smaller type

- When assigning a variable of larger type to a variable of smaller type, the least significant bits will be lost !
- The ”larger variable” is temporarily regarded as having the type of the ”smaller variable”.



Test (”socrative.com”: Room = MSYS)

- What is the value of z after this:

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

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

Example

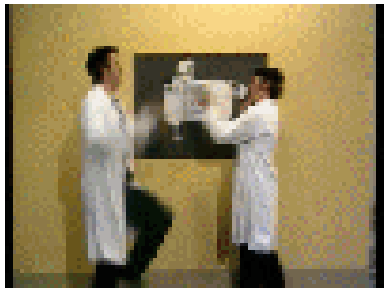
```
unsigned long z;
unsigned long x = 15000;
unsigned int y = 63000;

// What is the problem here ?
z = x + (10 * y);
```

- **Problem: The type of y (16 bits) is to small for storing the ”intermediate result” (10 * y).**
- **If y had the type of an unsigned long (32 bits) the problem would have been solved.**
- **Since we only have the problem for the ”intermediate result” , we can solve this by using ”Type Casting”.**

Type casting

- ”Type Casting” a variable is done by writing the desired type in brackets in front of the variable.

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

// Now the compiler will treat the
// ”intermediate result” (10 * y) as 32 bits.
// - and the result will be correct.
// But still y is 16 bits in memory.
z = x + (10 * (unsigned long)y);
```



C and the ports

Example 7-1

Write an AVR C program to send values 00–FF to Port B.

Solution:

```
#include <avr/io.h>                    //standard AVR header

int main(void)
{
   unsigned char z;
   DDRB = 0xFF;                        //PORTB is output
   for(z = 0; z <= 255; z++)
      PORTB = z;

   return 0;
}
//Notice that the program never exits the for loop because if you
//increment an unsigned char variable when it is 0xFF, it will
//become zero.
```

C and the ports

Example 7-4

Write an AVR C program to send values of –4 to +4 to Port B.

Solution:

```
#include <avr/io.h>                    //standard AVR header

int main(void)
{
   char mynum[ ]  = { -4,-3,-2,-1,0,+1,+2,+3,+4} ;
   unsigned char z;

   DDRB = 0xFF;                        //PORTB is output

   for(z=0; z<=8; z++)
     PORTB = mynum[ z] ;

   while(1);                           //stay here forever
   return 0;
}
```

Run the above program on your simulator to see how PORTB displays values of FCH, FDH, FEH , FFH, 00H, 01H, 02H, 03H, and 04H (the hex values for –4, –3, –2, –1, 0, 1, etc.). See Chapter 5 for discussion of signed numbers.

16 bit loop counter

Example 7-5

Write an AVR C program to toggle all bits of Port B 50,000 times.

Solution:

```
#include <avr/io.h>                    //standard AVR header
int main(void)
{
   unsigned int z;
   DDRB = 0xFF;                        //PORTB is output

   for(z=0; z<50000; z++)
   {
      PORTB = 0x55;
      PORTB = 0xAA;
   }

   while(1);                           //stay here forever
   return 0;
}
```

Run the above program on your simulator to see how Port B toggles continuously. Notice that the maximum value for unsigned int is 65,535.

Time delays in C (“primitive method”)

- You can use the for to implement a time delay :

```
void delay100ms(void){

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

Disadvantages using this method :

- The delay will depend on clock frequency ☹
- The delay will depend fully on the compiler ☹
- The level of optimatization MUST be O0 (“none”) ☹

Primitive delays method

Example 7-7

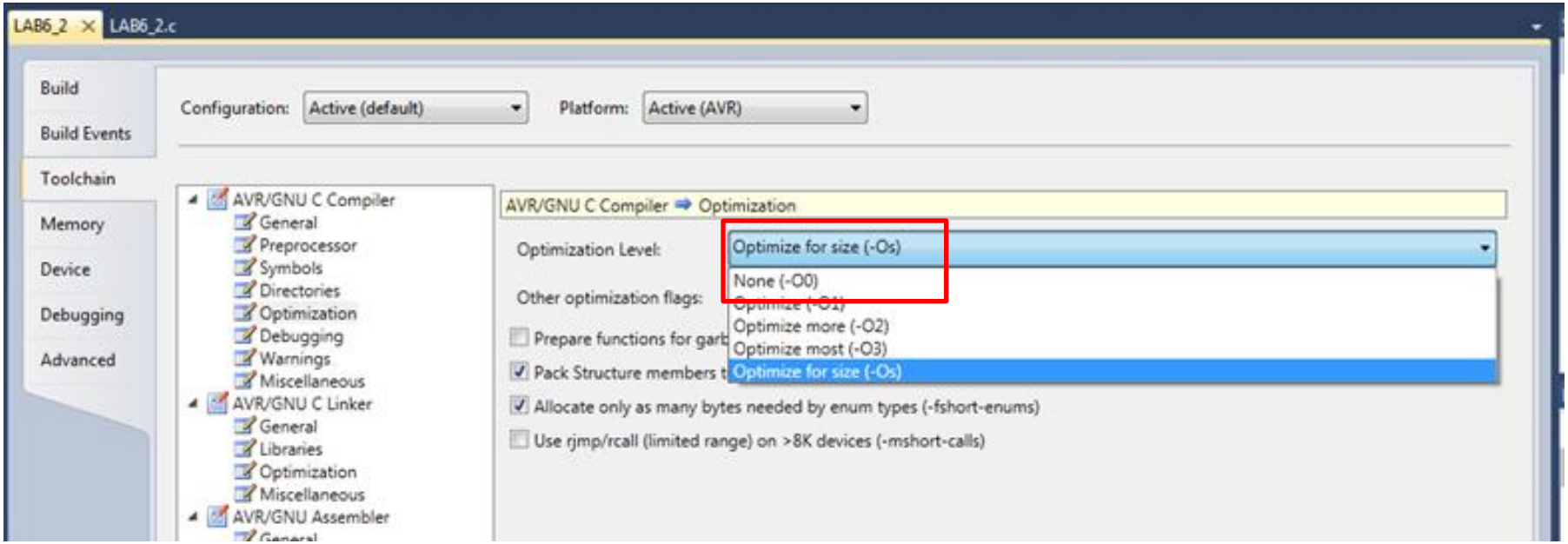
Write an AVR C program to toggle all the bits of Port B continuously with a 100 ms delay. Assume that the system is ATmega 32 with XTAL = 8 MHz.

Solution:

```
#include <avr/io.h>                    //standard AVR header
void delay100ms(void)
{
   unsigned int i;
   for(i=0; i<42150; i++);             //try different numbers on your
}                                      //compiler and examine the result.

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

NB: Compiler optimization has to be ”none” !
Why ?

How to disable optimization “level O0”



Select ”Properties” for the project.
Under ”Toolchain” → ”Optimization” select ”None”.

BUT then in-effektive code is generated ☹

Time delay functions in C

- BETTER:
  Use pre-defined compiler functions for generating time delays.

Start with :

```
#define F_CPU 3686400
#include <util/delay.h>
```

- and then you can use the functions:

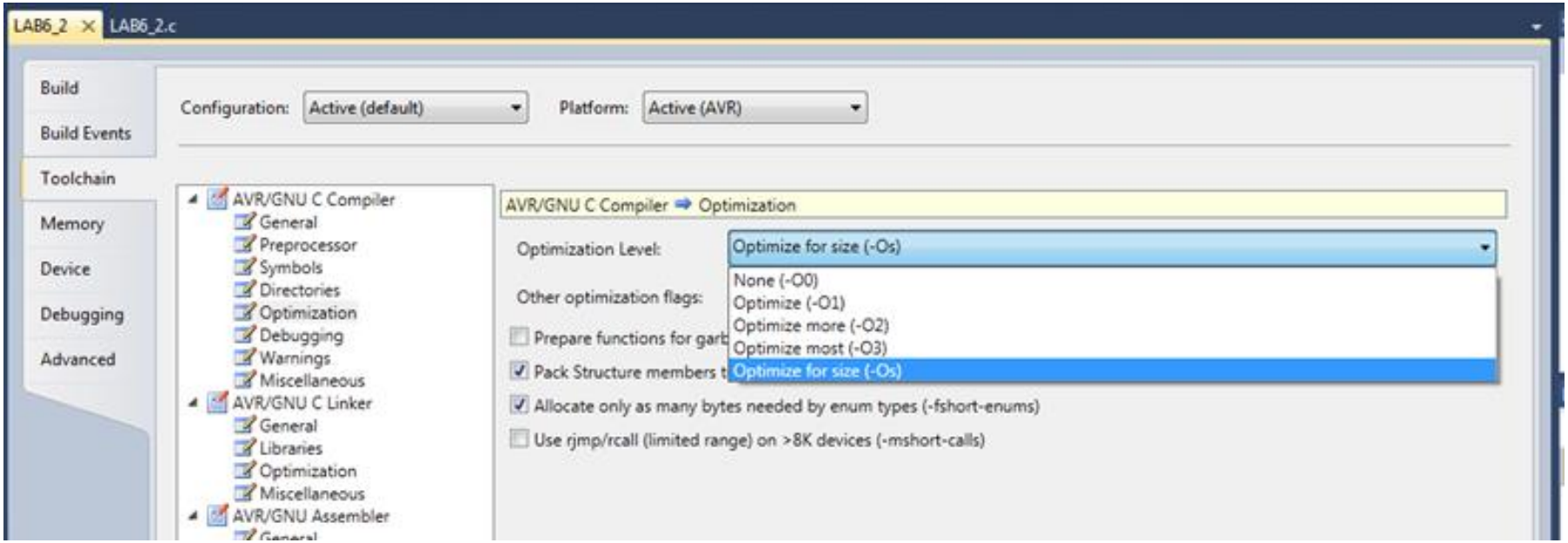
```
_delay_ms(1000);
_delay_us(1000);
```

Compiler optimization and ”delay.h”

IMPORTANT:

Using the delay library <util/delay.h>, the compileren optimization must not be ”none”.

This is illogical, but still true.



AVR GCC delay functions

Example 7-8

Write an AVR C program to toggle all the pins of Port C continuously with a 10 ms delay. Use a predefined delay function in Win AVR.

Solution:

```
#include <util/delay.h>                  //delay loop functions
#include <avr/io.h>                      //standard AVR header

int main(void)
{
      void delay_ms(int d)               //delay in d microseconds
      {
            _delay_ms(d);
      }
      DDRB = 0xFF;                       //PORTA is output
      while (1){
            PORTB = 0xFF;
            delay_ms(10);
            PORTB = 0x55;
            delay_ms(10);
      }
      return 0;
}
```

The example uses a ”wrapper function”. This is NOT necessary.

Logical operators

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
```

Logical operators in 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	

Example 7-12

Run the following program on your simulator and examine the results.

```
#include <avr/io.h>              //standard AVR header
int main(void)
{
   DDRB = 0xFF;                  //make Port B output
   DDRC = 0xFF;                  //make Port C output
   DDRD = 0xFF;                  //make Port D output
   PORTB = 0x35 & 0x0F;          //ANDing
   PORTC = 0x04 | 0x68;          //ORing
   PORTD = 0x54 ^ 0x78;          //XORing
   PORTB = ~0x55;                //inverting
   while (1);
   return 0;
}
```

Logical operators

Example 7-13

Write an AVR C program to toggle only bit 4 of Port B continuously without disturbing the rest of the pins of Port B.

Solution:

```
#include <avr/io.h>                    //standard AVR header

int main(void)
{
   DDRB = 0xFF;                        //PORTB is output

   while(1)
   {
      PORTB = PORTB | 0b00010000;      //set bit 4 (5th bit) of PORTB
      PORTB = PORTB & 0b11101111;      //clear bit 4 (5th bit) of PORTB
   }

   return 0;
}
```

Logical operators

Example 7-14

Write an AVR C program to monitor bit 5 of port C. If it is HIGH, send 55H to Port B; otherwise, send AAH to Port B.

Solution:

```
#include <avr/io.h>              //standard AVR header

int main(void)
{
   DDRB = 0xFF;                  //PORTB is output
   DDRC = 0x00;                  //PORTC is input
   DDRD = 0xFF;                  //PORTB is output

   while(1)
   {
      if (PINC & 0b00100000)     //check bit 5 (6th bit) of PINC
        PORTB = 0x55;
      else
        PORTB = 0xAA;
   }

   return 0;
}
```

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 in 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 in 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 */
```

What assembly instruction correspons to >> ?
What assembly instruction correspons to << ?

Using 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)`

How to set (to 1) a bit in a byte

- We can use the | operator to set a bit in a byte to 1

```
    xxxx xxxx
|   0001 0000
    ----------
    xxx1 xxxx
```

OR

```
    xxxx xxxx
|   1 << 4
    ----------
    xxx1 xxxx
```

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

How to reset (to 0) a bit in a byte

- We can use the & operator to reset a bit in a byte

```
      xxxx xxxx
&     1110 1111
      ----------
      xxx0 xxxx
```

OR

```
      xxxx xxxx
&   ~(1 << 4)
      ----------
      xxx0 xxxx
```

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

How to check a bit in a byte

- We can use the & operator to check, if a bit in a byte is 1 or 0 :

```
      xxxx xxxx
&     0001 0000
      ----------
      000x 0000
```

OR

```
      xxxx xxxx
&   (1 << 4)
      ----------
      00x0 0000
```

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

REMEMBER the C rule :

- ”Everything being 0” = FALSE.
- ”Everything NOT being 0” = TRUE.

Bit shifting

Example 7-23

Write an AVR C program to monitor bit 7 of Port B. If it is 1, make bit 4 of Port B input; else, change pin 4 of Port B to output.

Solution:

```
#include <avr/io.h>                          //standard AVR header

int main(void)
{
  DDRB = DDRB & ~(1<<7);                     //bit 7 of Port B is input

  while (1)
  {
    if(PINB & (1<<7))
      DDRB = DDRB & ~(1<<4);                 //bit 4 of Port B is input
    else
      DDRB = DDRB | (1<<4);                  //bit 4 of Port B is output
  }

  return 0;
}
```

Bit shifting

Example 7-24

Write an AVR C program to get the status of bit 5 of Port B and send it to bit 7 of port C continuously.

Solution:

```
#include <avr/io.h>                    //standard AVR header

int main(void)
{
   DDRB = DDRB & ~(1<<5);              //bit 5 of Port B is input
   DDRC = DDRC | (1<<7);               //bit 7 of Port C is output

   while (1)
   {
      if(PINB & (1<<5) )
         PORTC = PORTC | (1<<7);       //set bit 7 of Port C to 1
      else
         PORTC = PORTC & ~(1<<7);      //clear bit 7 of Port C to 0
   }
   return 0;
}
```

#define for bit numbers

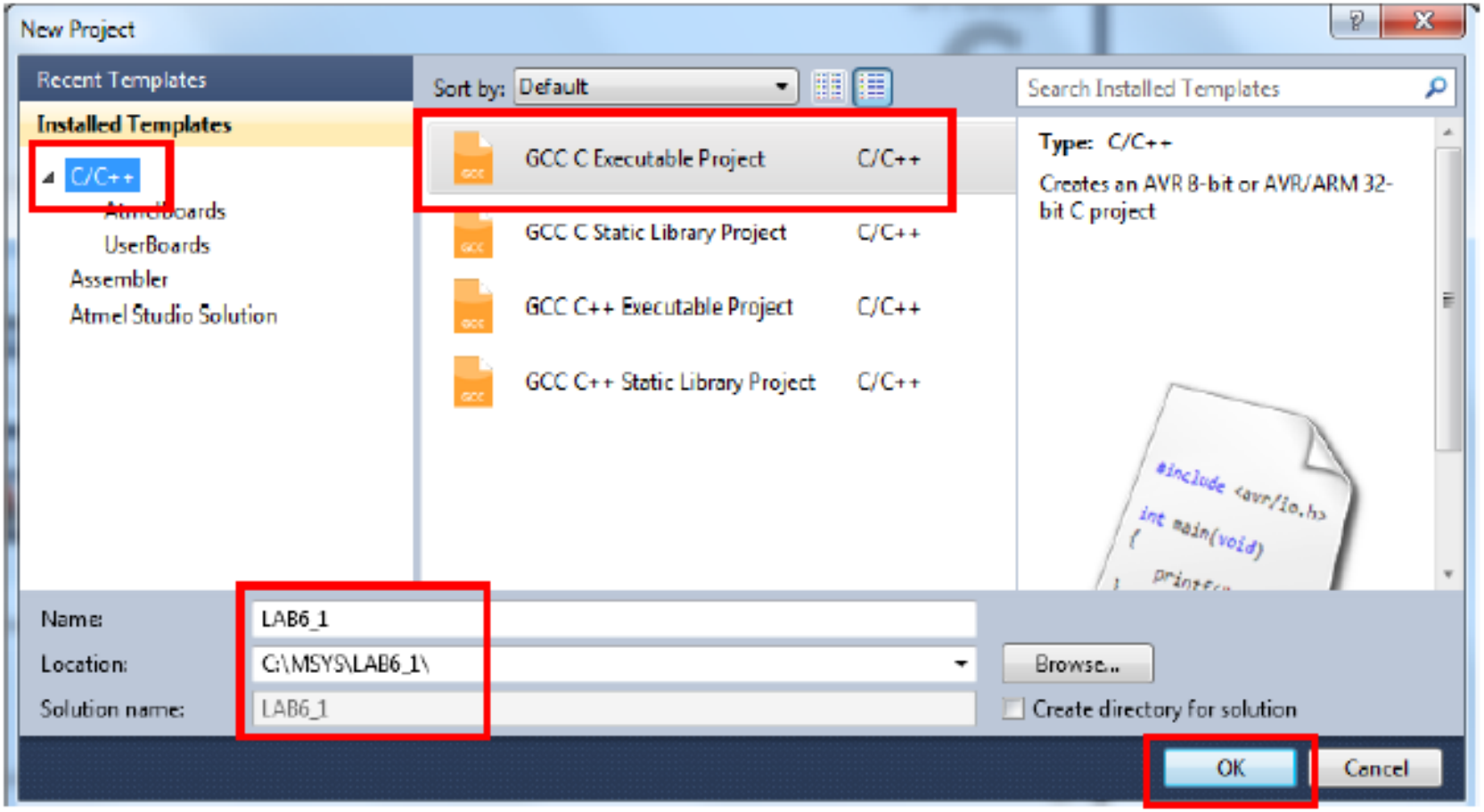
Example 7-25

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.

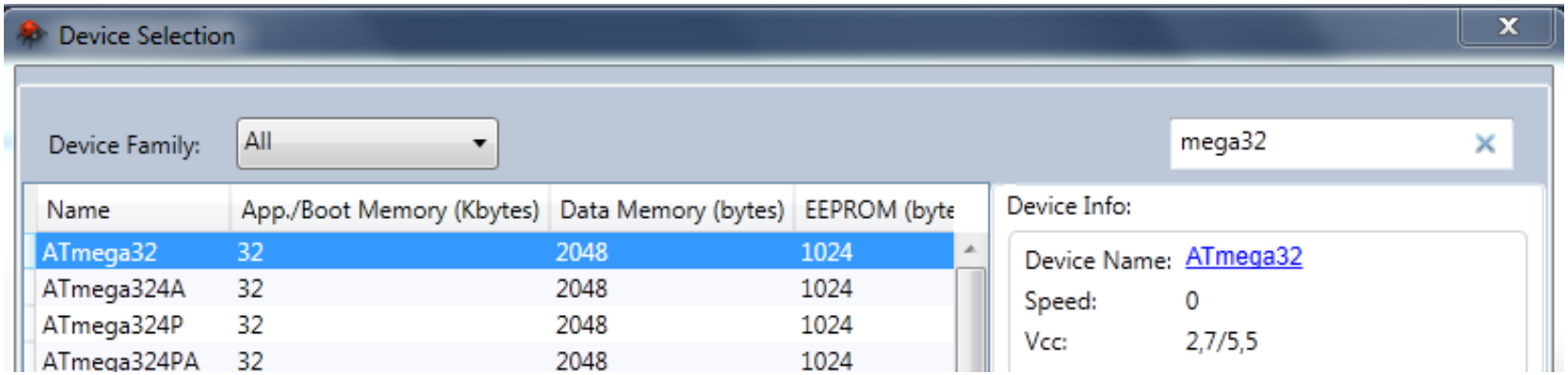
Solution:

```
#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 projekt in Atmel Studio 6



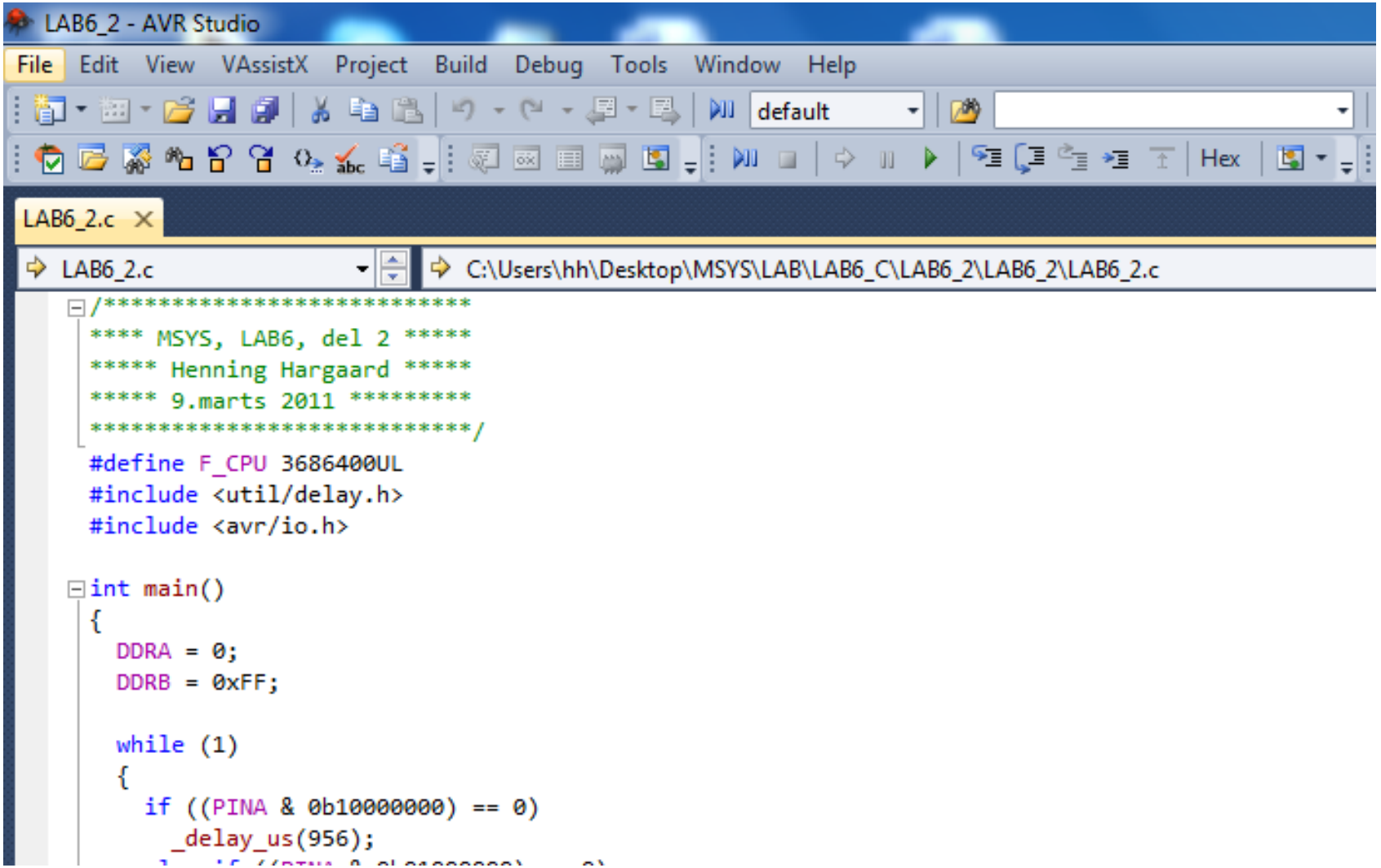
ACR GCC project in Atmel Studio 6



LAB6_1.c

```
/*
 * LAB6_1.c
 *
 * Created: 21-09-2011 14:05:46
 *  Author: hh
 */

#include <avr/io.h>

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

ACR GCC project in Atmel Studio 6



End of lesson 10