C++ mit dem Arduino (Uno und Mega2560)

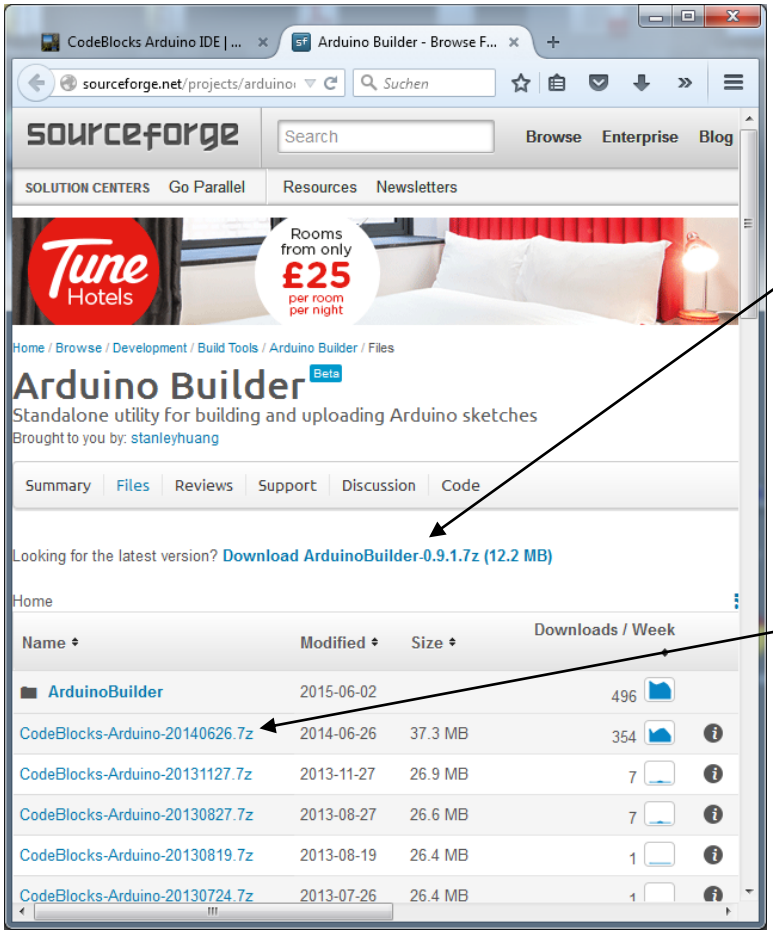
1. Toolchain

Als toolchain bezeichnet man die Reihe von Software, die nötig ist um den Prozessor zu programmieren, das Ergebnis draufzuladen und dann zu testen.

Hier sehr simpel :

Im Hinblick auf die Objektorientierung, und weil das schön klein und übersichtlich ist, verwenden wir Code::Blocks. Laden Sie die schon für den Arduino optimierte Version : http://arduinodev.com/codeblocks/

Wenn man den Download-Buttons folgt, kommt man zur Quelle sourceforge.net, dort zur einer solchen Seite :



Falscher Link : Das ist nur das Tool zum Laden auf den Controller

CodeBlocks-Arduino brauchen Sie !

Laden, installieren, und den Arduino über USB an den PC anschließen. Versorgungsspannung braucht er keine, das geht praktischerweise über USB.

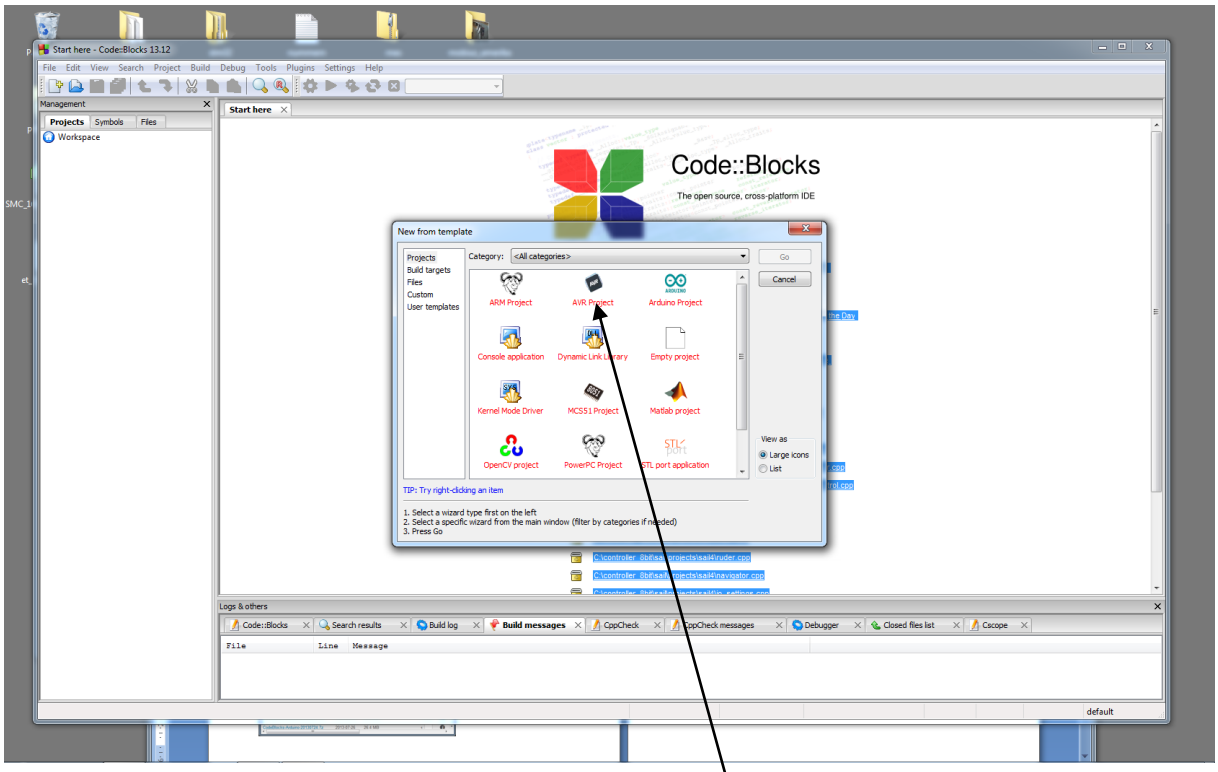
In Systemsteuerung/System/Gerätemanager sollte unter „COM und LPT“ der Arduino als Beschaltung an einem neuen COM-Port erscheinen (z.b. COM3)

Wenn das nicht klappt, liegt es wahrscheinlich an der Emulation des COM-Ports über USB. Vielleicht einen Treiber laden ? -> Google !

2. Bedienung

Im installierten Verzeichnis finden Sie die Code::Blocks – exe. (Das Programm mit dem Code::Blocks- Grafiksymbol).

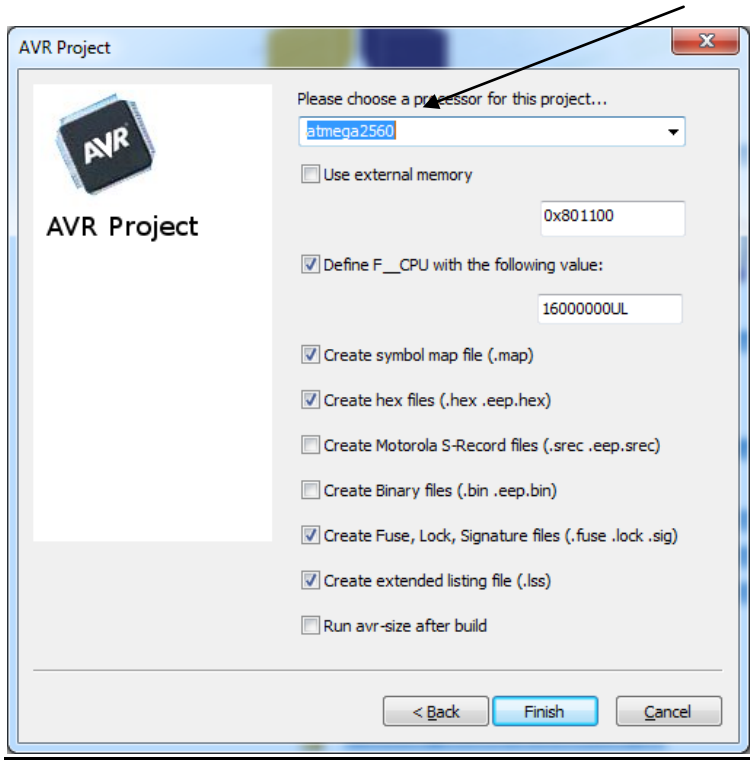
Öffnen .. dann auf File-> New-> Project : (Name egal)



Wir arbeiten mit einem AVR-Prozessor
(nicht mit der Ardunio-Umgebung !)

Die weiteren Fenster klicken wir ohne Änderung durch, der zu vergebende Projektname ist beliebig.

Bei der Auswahl des Prozessors muß der Typ gewählt werden, der auf unserem Board eingebaut ist. Beim Mega ist es ein ATMega2560 :



Der Rest bleibt unverändert.

2. Dokumentation

Der Arbeitsvorgang ist eigentlich immer der gleiche, egal was sie mit dem Prozessor machen wollen. Sie lesen im Handbuch (reference manual) nach, wie man mit dem jeweiligen Teil der Prozessorhardware umzugehen hat, also Ein-Ausgänge, die serielle Schnittstelle, der I2C-Bus, die Timer, usw…
Dann brauchen sie noch den Schaltplan (schematic), um nachzuschauen, an welchen Pins die jeweilige Funktion angeschlossen ist. Im Fall des Arduino müssen sie dann noch nachschauen, welcher Pin auf der Arduino-Platine an welchem Prozessorpin hängt (mapping). Diese Dokus finden Sie hier :

Arduino Uno

Datasheet : http://www.atmel.com/images/doc8161.pdf

Pinmapping : https://www.arduino.cc/en/Hacking/PinMapping168

Arduino Mega2560

Datasheet : http://www.atmel.com/images/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf

Pinmapping : https://www.arduino.cc/en/Hacking/PinMapping2560

3. theoretische Vorbereitung

Ein erstes Beispiel, barrierefrei aufbereitet :

Eine LED soll blinken. Hierzu sind die Seiten im Handbuch wichtig, die sich mit I/O – Pins beschäftigen. Wird manchmal auch GPIO genannt: general purpose input output.
Im Inhaltsverzeichnis (Uno : Seite 443, Mega : Seite 427) zu finden unter „Ports as General Digital I/O“.
Das ist dort sehr umfassend erklärt, nur in Spezialfällen braucht man auch die innerer Hardwarebeschaltung.

Für uns langt im Prinzip aber (beim Uno) Bild 13-1 auf Seite 77 bzw. beim Mega auf Seite 69. Da steht im Prinzip, daß ein Pin Eingang wird, wenn man das ***Datenrichtungsregister*** des Pin auf Null legt, und Ausgang, wenn man es auf 1 legt. Wie das Register zu beschreiben ist, steht beim Uno auf Seite 92, zum Beispiel 13.4.3, die Definition der Datenrichtungen für die Pins von Port B.

Beim Mega suchen sie auf Seite 96.

Wir suchen jetzt ein Pin eines beliebigen Ports am Arduino aus.

Das machen Sie jetzt selber : welche Prozessorports sind denn auf der Arduino-Leiterplatte auf Pins rausgeführt ? Einen suchen und notieren (oder können sie sich sowas merken ?). Das Datenrichtungsregister suchen. Welches Bit muß 1 (=Ausgang) werden ?

4. Realisierung

Ich wähle „digital pin 7“ auf dem Arduino UNO (leicht zu finden auf der Platine), oder beim Mega vielleicht die auf der Platine fest installierte LED an PORT B7, dann muß ich nicht basteln (sowas gibt’s auf dem Uno auch : suchen !)

- Weiter im Beispiel jetzt mit dem UNO :

Nachschauen im mapping : das ist PD7 vom Prozessor, also Pin 7 vom PortD
Also muß das Datenrichtungsregister von PD7 auf 1 … das ist Bit 7 von DDRD.

Dann muß man in vernünftigen Zeitabständen den PD7 zwischen 1 und 0 wechseln, damit das blinkt. Die Zeit machen wir mit einer Library von avr, siehe unten.

- Oder mit dem Mega :

PB7 ist Bit 7 von PortB, also muß Bit 7 vom DDRB 1 (=Ausgang) werdden

Hardware : eine Reihenschaltung von LED und einem Widerstand (rund 1k) vom Pin nach GND einbauen, wenn das Pin nicht schon an ner LED hängt.

Zum 1 oder 0 schalten ein wenig Digitaltechnik (eigentlich hier Voraussetzung;-) :

Ein Bit in einem Byte kann man auf verschiedene Arten setzen :

1) Entweder das ganze Byte auf einen Sitz mittels einer Dualzahl :
Hier z.b. 128 in DDRD schreiben, das setzt (siehe Binärzahl) alles 0 bis auf Bit 7.
Ist sehr unelegant, weil wir die andereen Bits ja gar nicht beeinflussen wollen …

2) Mit einem logischen ODER eine 1 an die richtige Stelle :
DDRD = DDRD | 128; (das | macht das ODER für alle 8 Bit, Bit 7 wird damit 1)

3) In C geht das auch mit |= bzw. mit &= ~ , suchen sie mal ….

5. Programm

In Code::Blocks ein neues Projekt anlegen, und zwar eines vom Typ AVR.
Richtigen Prozessor wählen (zur Not auf dem Prozessor nachschauen).

Der Code für den UNO : (In die main.c reinschreiben)

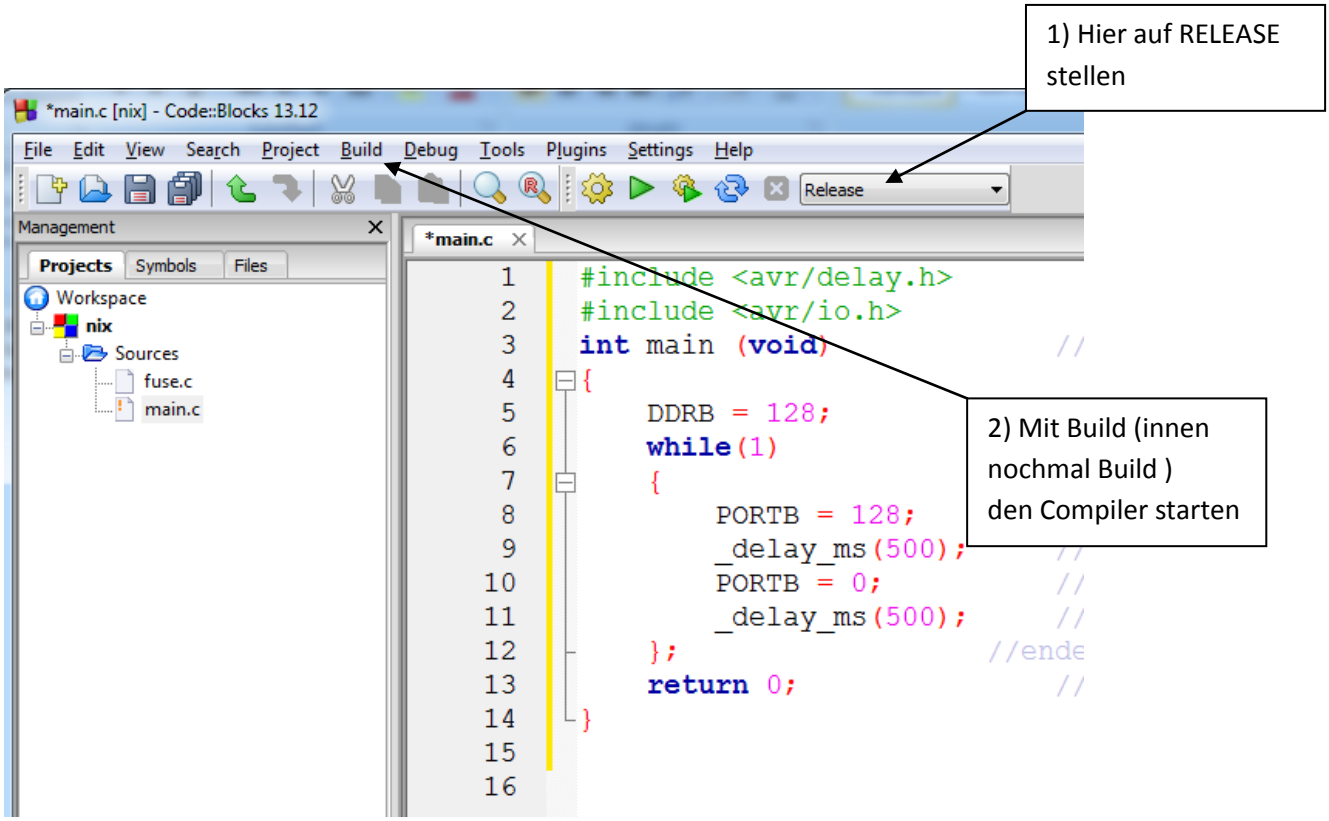
```
#include <avr/delay.h>          //nach <avr kommt im Editor ein Menü
#include <avr/io.h>

int main (void)                 //hier geht das Hauptprogramm los
{
        DDRD = 128;             //Port initialisieren
        while(1)                //was jetzt kommt läuft endlos
        {
                PORTD = 128;    //led an
                _delay_ms(500); //warten
                PORTD = 0;      //led aus
                _delay_ms(500); //warten
        };                      //ende der while(1)-schleife
        return 0;               //C-Konvention : kein Rückgabewert
}                               //ende des hautpprogramms
```

Für den Mega diesen Code (anderer Ausgangspin) :

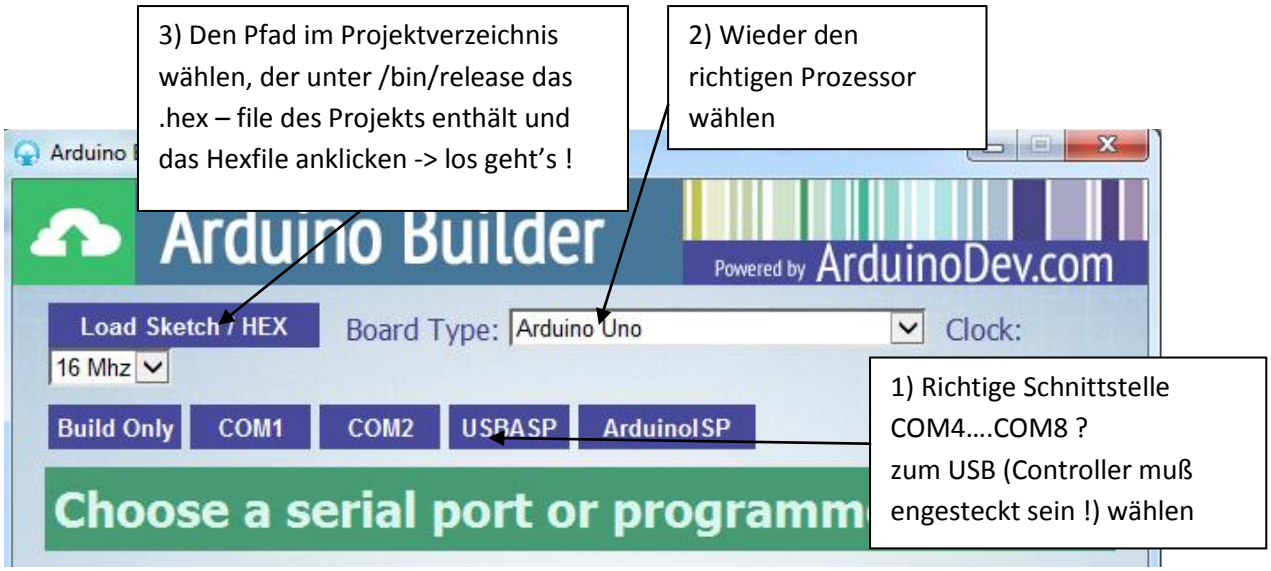
```
#include <avr/delay.h>                  //nach <avr kommt im Editor ein Menü
#include <avr/io.h>

int main (void)                         //hier geht das Hauptprogramm los
{
        DDRB = 128;                     //Port initialisieren
        while(1)                        //was jetzt kommt läuft endlos
        {
                PORTB = 128;            //led an, sehr unelegant gelöst
                _delay_ms(500);         //warten
                PORTB = 0;              //led aus
                _delay_ms(500);         //warten
        };                              //ende der while(1)-schleife
        return 0;                       //C-Konvention : kein Rückgabewert
}
```



Nach dem Compilerlauf währen unten im Status „0 ERRORS“ prima.
Warnings kann man zur Not erst mal ignorieren ….

Jetzt unter Tools den ARDIUNO BUILDER starten :



Jetzt blinkt’s, oder ?