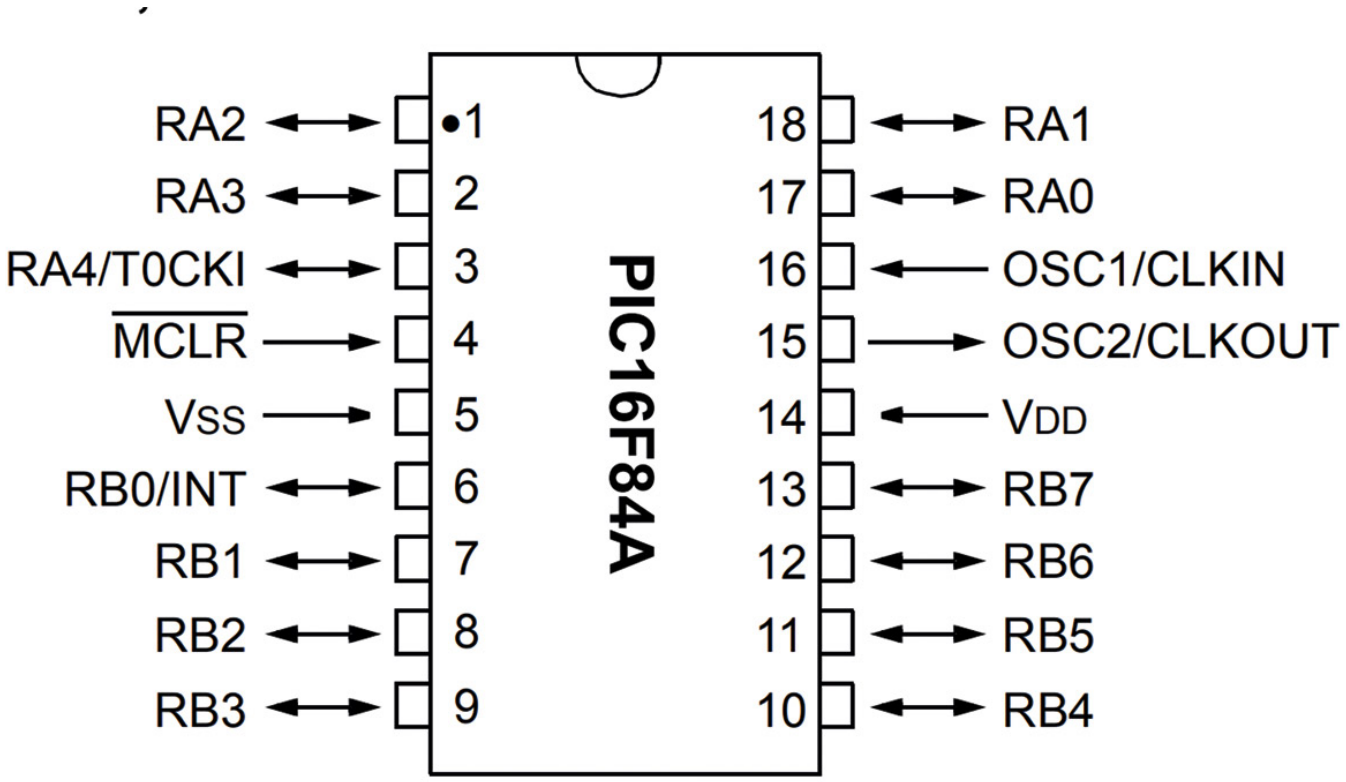
Einsteig in PIC Programmierung

Die Firma Microchip hat mit den PIC-Mikrocontrollern eine ganze Familie von leistungsfähigen Bausteinen entwickelt. In diesem Artikel sehen wir uns den PIC16F84A etwas genauer an. Wir schreiben ein einfaches Testprogramm in C und bringen es in den PIC-Controller. Als Entwicklungsumgebung verwenden wir die MPLAB X IDE.

Der PIC16F84A ist ein Mikrocontroller mit einer 8-Bit-RISC-(Reduced-Instruction-Set-Computing-)CPU. Der Hersteller hat das Reduzieren sehr ernst genommen, die CPU versteht gerade einmal 35 Befehle. Im Vergleich dazu hat ein Intel-i7-Prozessor weit über 300 Befehle. Die 35 Befehle sind so aufgebaut, dass sie alle innerhalb eines Taktzyklus ausgeführt werden können. Wenn man bedenkt, dass der maximale Takt bei 20 MHz liegt, ist das eine ganze Menge. Die Betriebsspannung des PIC16F84A kann zwischen 2 und 5,5 Volt liegen. Der Speicherausbau des Controllers wirkt auf den ersten Blick etwas klein (68 Byte RAM, 64 Byte EEPROM, 1024 Byte Programmspeicher), reicht aber für viele Anwendungsfälle vollkommen aus.

Die 13 programmierbaren IO Pins können bis zu 25 Milliampere aufnehmen oder liefern. Dadurch ist der PIC16F84A bestens für kleine Automatisierungsaufgaben geeignet. Die Familie der PIC-Controller ist groß und es gibt natürlich auch leistungsfähigere Controller mit mehr IOs oder auch mit analogen Ein- und Ausgängen. Auf der Herstellerseite erhalten Sie einen Überblick über die einzelnen Modelle.

Erwerben können Sie den PIC16F84A für unter 4 Euro. Der PIC16F84A wird in einem 18-Pin-DIL-Gehäuse verbaut. Auf nachstehende Abbildung ist das Pin-out des ICs zu sehen.

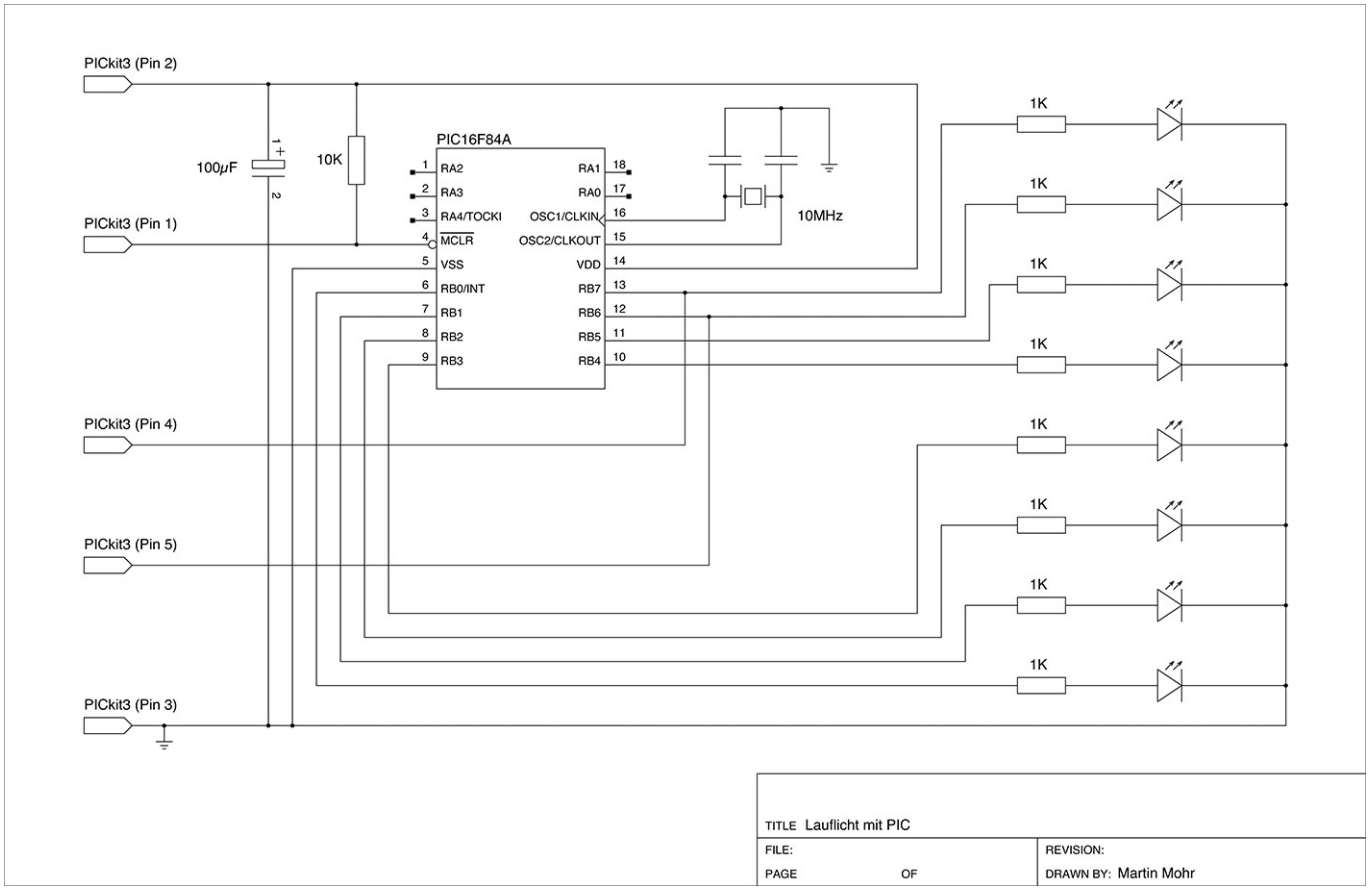


IDE, Programmieradapter und Testhardware

Wie eingangs erwähnt, verwenden wir zum Programmieren des PIC-Controllers die MPLAB X IDE. Sie kann frei von der Microchip-Homepage [4] heruntergeladen werden. Die IDE steht für alle gängigen Betriebssysteme bereit. Unter Linux benötigen Sie allerdings Root-Rechte, um die Installation zu starten. Die Installation ist komplett durch einen Wizard gesteuert und lässt sich ohne Probleme durchführen. Zusätzlich zur IDE benötigen wir noch den zu unserem Mikrocontroller passenden Compiler, der ebenfalls frei von der Microchip-Homepage [5] heruntergeladen werden kann. Für den PIC16F84A benötigen Sie den MPLAB XC8 Compiler v2.10. Die Installation funktioniert nach dem gleichen Muster wie bei der IDE. Die Trennung von IDE und Compiler hat den Vorteil, dass man alle Mikrocontroller des Herstellers mit nur einer IDE programmieren kann. Man muss sich also nicht für jeden neuen Controller-Typ an eine andere IDE gewöhnen. Die MPLAB X IDE integriert die üblichen Versionsverwaltungstools wie SVN, Git usw.

Um die Bausteine der PIC-Familie programmieren zu können, benötigen Sie einen speziellen Programmieradapter. Für diesen Artikel wurde ein Clone des PICkit 3 verwendet. Man kann ihn für ca. 10 Euro im Handel erwerben. Der Original-PICkit-3-Programmieradapter kostet dagegen um die 60 Euro.

Den Schaltplan für unseren Testaufbau können Sie auf der nachstehende Abbildung sehen. Der PIC16F84A ist hier mit dem PICkit 3 verbunden. Der Kondensator ist nötig, weil beim Programmieren des Mikrocontrollers sonst die Betriebsspannung des Programmieradapters zusammenbricht. Wenn Sie die Schaltung über ein zusätzliches Netzteil mit Energie versorgen, ist er nicht nötig.



An den Eingängen OSC1 und OSC2 ist ein 10-MHz-Quarz angeschlossen. Die zwei Kondensatoren am Quarz sorgen dafür, dass der Quarz exakt auf seiner Frequenz schwingt. Die konkreten Werte müssen

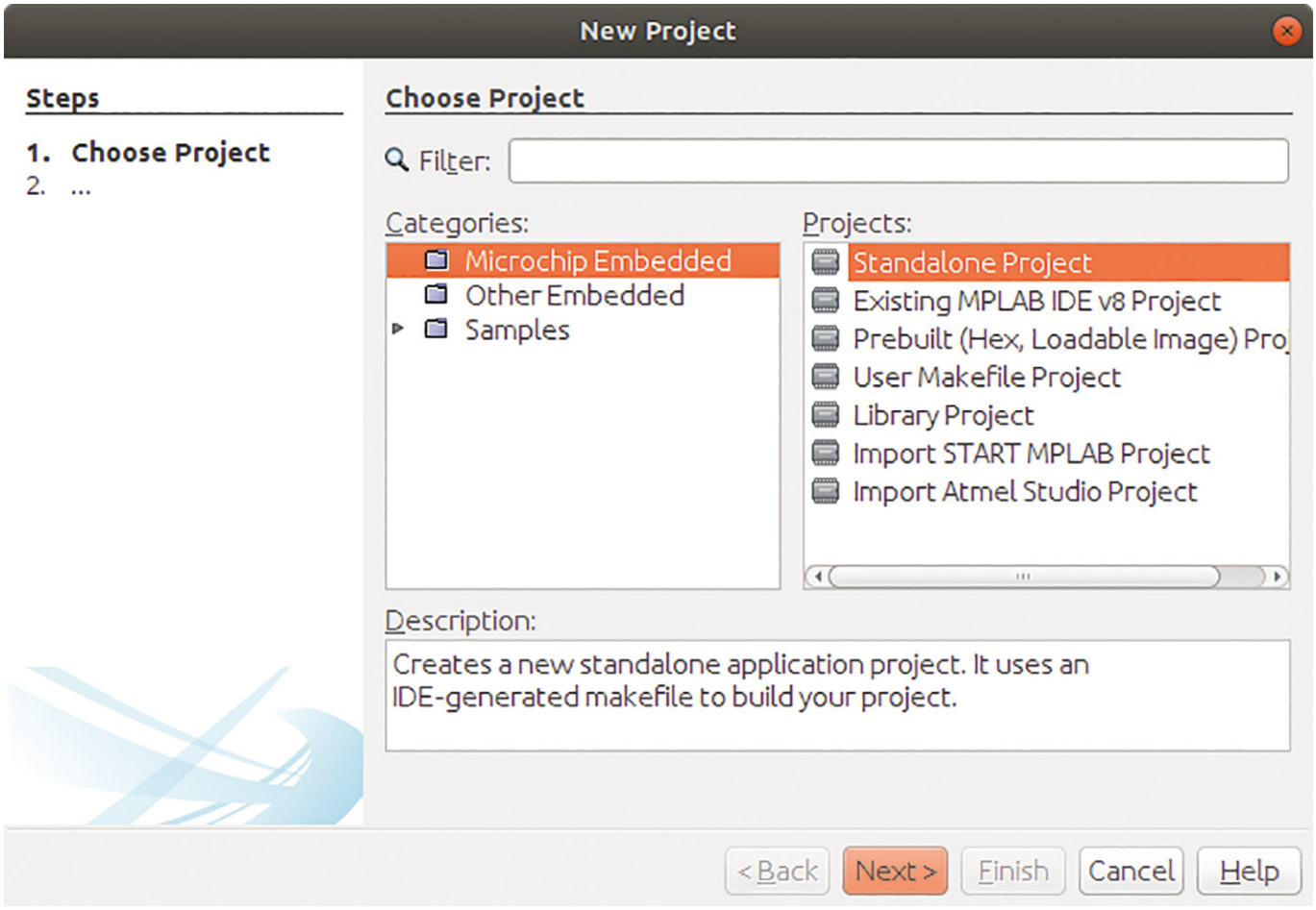
dem Datenblatt des Quarzes entnommen werden. Oft werden hier 22 pF eingesetzt, weil es immer so gemacht wird, das ist aber leider nur selten der richtige Wert. Der Quarz schwingt auch mit den falschen Kondensatoren, halt nur nicht exakt mit der richtigen Frequenz. Für jeden Typ gibt der Hersteller an, welche Kapazität die Kondensatoren haben müssen. Der 10-K-Widerstand ist nötig, damit der PIC nach dem Programmieren von alleine mit der Ausführung des Programms beginnt. Fehlt der Widerstand, denkt der Controller, dass er sich im Programmiermodus befindet und startet nicht.

An dem IO-Port B sind 8 LEDs mit Vorwiderständen angeschlossen. Damit der Testaufbau etwas aufgeräumter aussieht, wurde hier ein LED Bar Graph verwendet. Sie können aber auch einzelne LEDs verwenden.

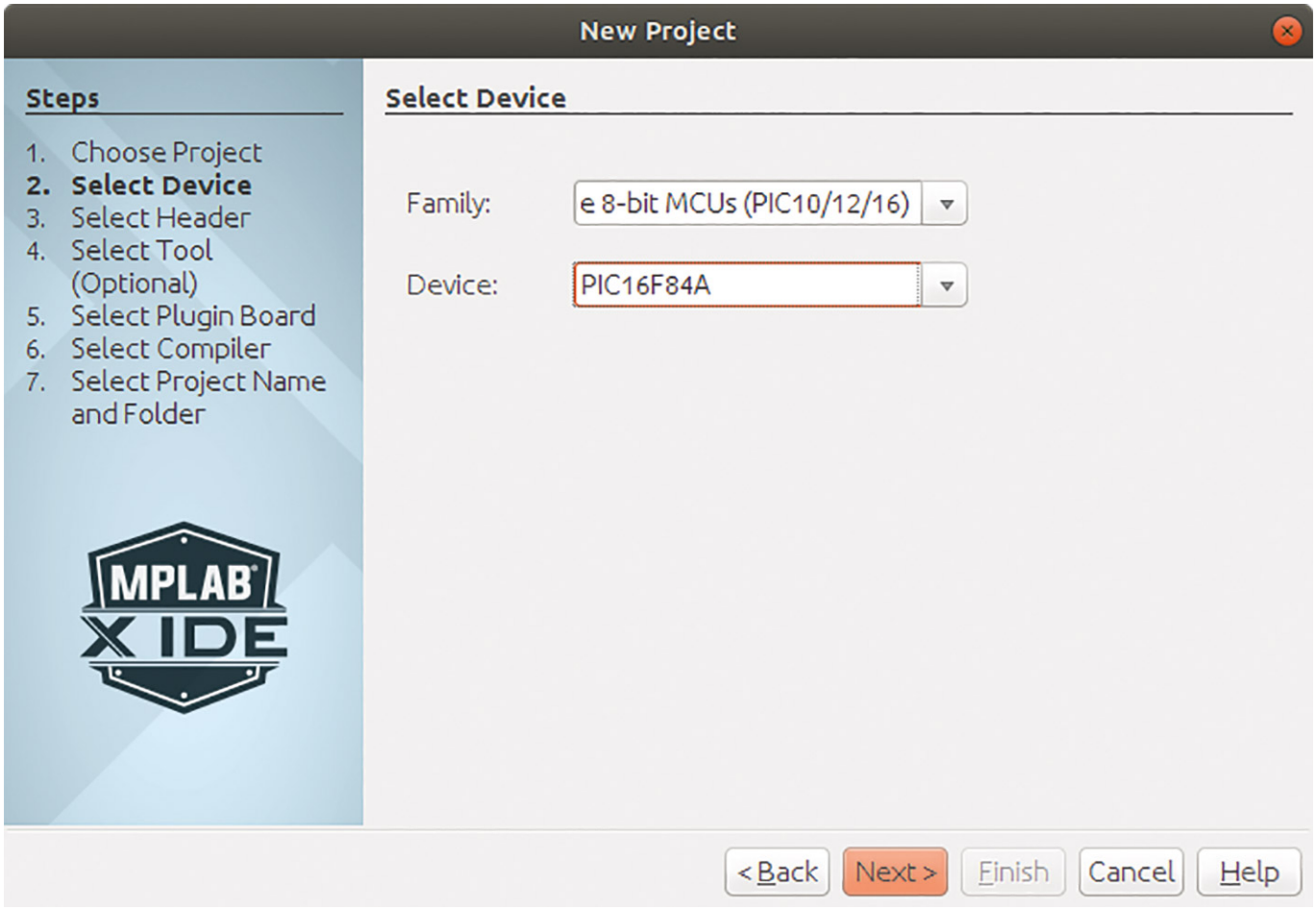
Projekt anlegen

Nun ist alles soweit vorbereitet, dass wir mit einem ersten kleinen Projekt starten können. Verbinden Sie bitte ihren PICkit-3-Programmieradapter mit einem USB-Port Ihres PCs. Danach können Sie die MPLAB X IDE starten.

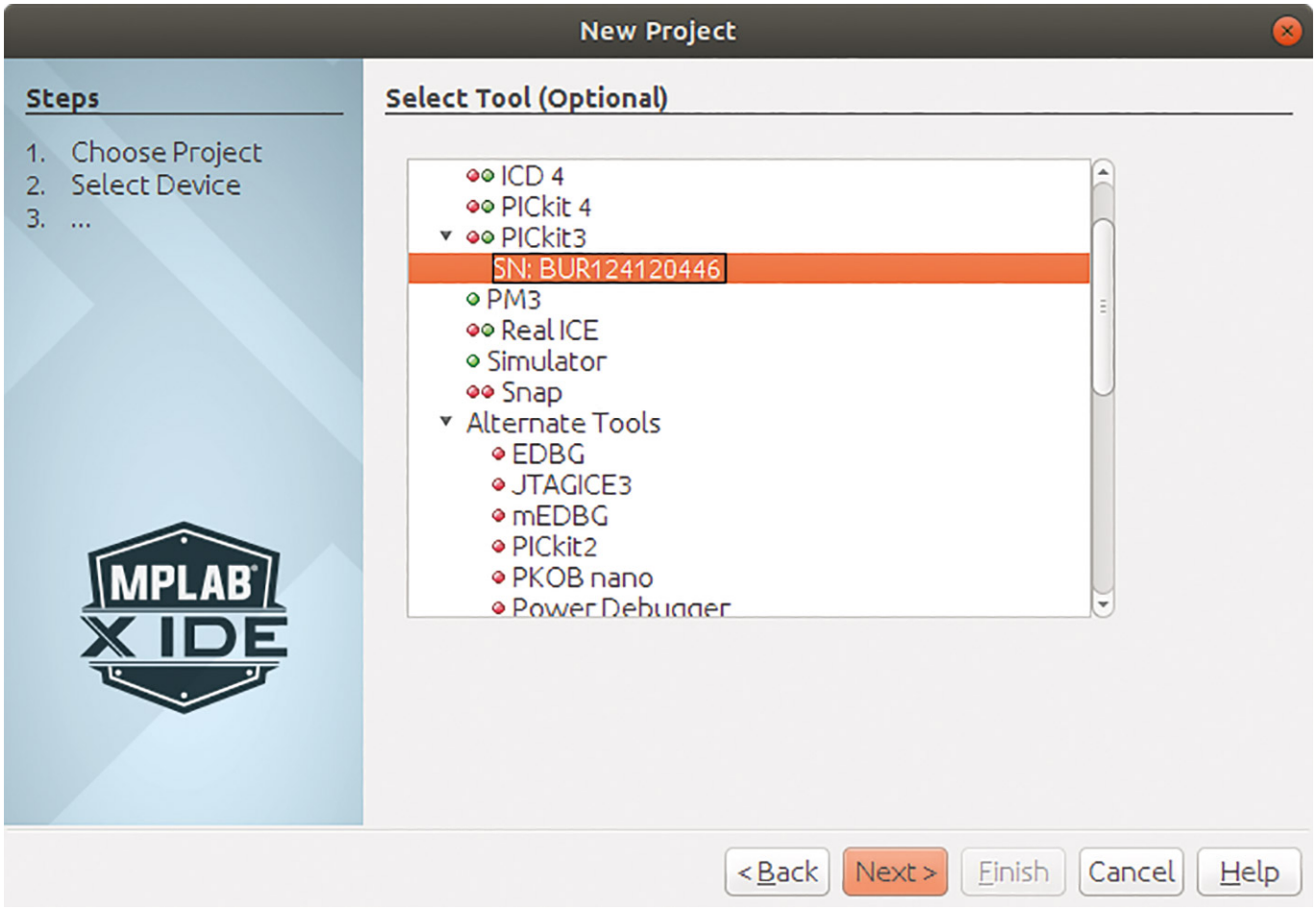
Unter dem Menüpunkt File | New Project legen wir nun ein neues Projekt an. Der Prozess ist komplett Wizard-gesteuert. Im ersten Fenster wählen wir den Projekttyp Microchip Embedded | Standalone Project aus und klicken dann auf Next-



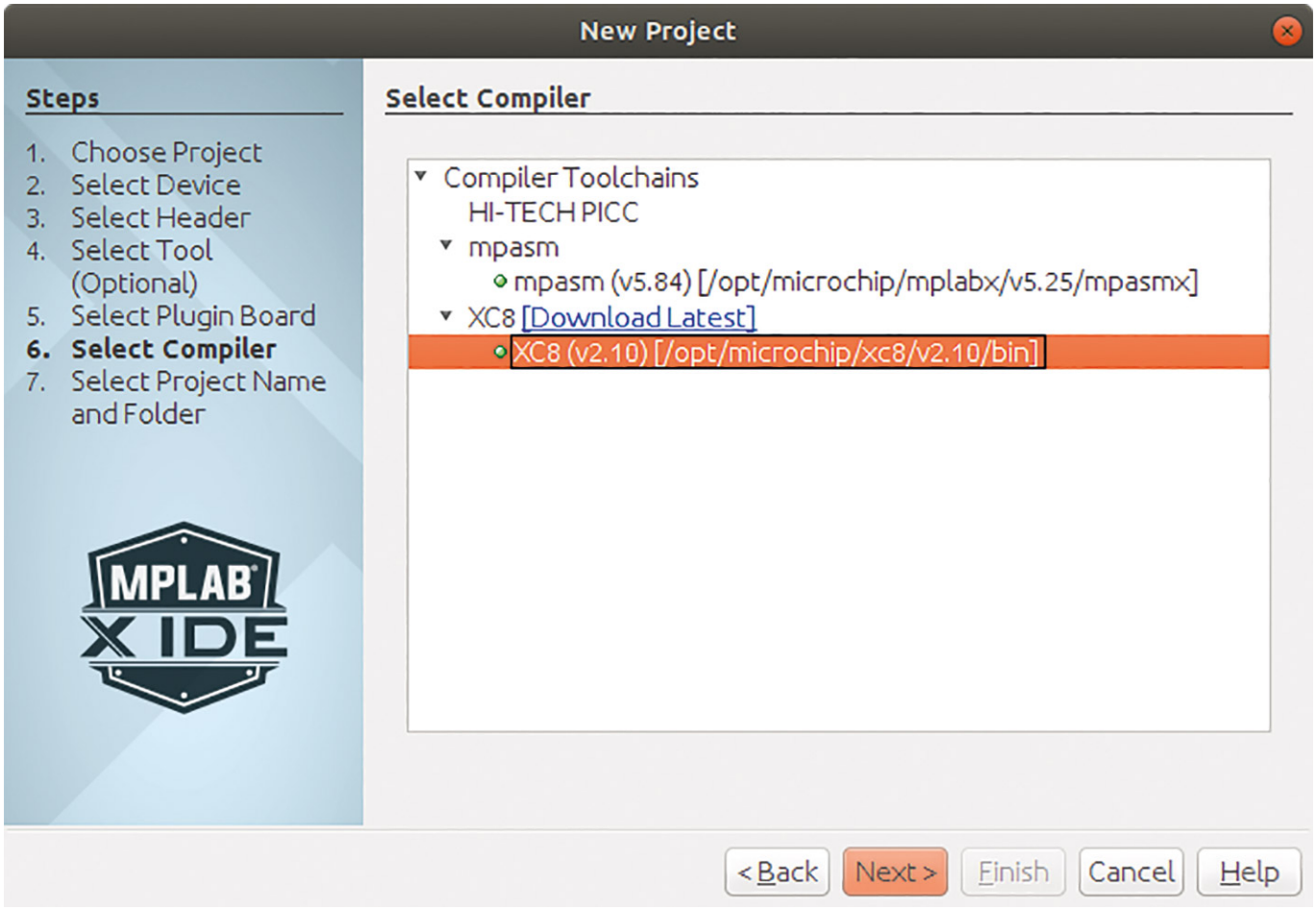
Auf der nächsten Seite des Wizard werden wir nach dem Gerät gefragt, für das wir ein Programm erstellen wollen. Als Familie wählen wir hier Baseline 8-bit MCUs (PIC10/12/16), danach suchen wir das Gerät PIC16F84A. Um zur nächsten Seite zu gelangen, klicken Sie auf Next.



Nun wird nach dem Programmieradapter, der verwendet werden soll, gefragt. Falls Ihr PICkit3 schon am USB-Port steckt, wird er direkt erkannt. Wählen Sie nun Ihre Seriennummer aus und klicken Sie auf Next.

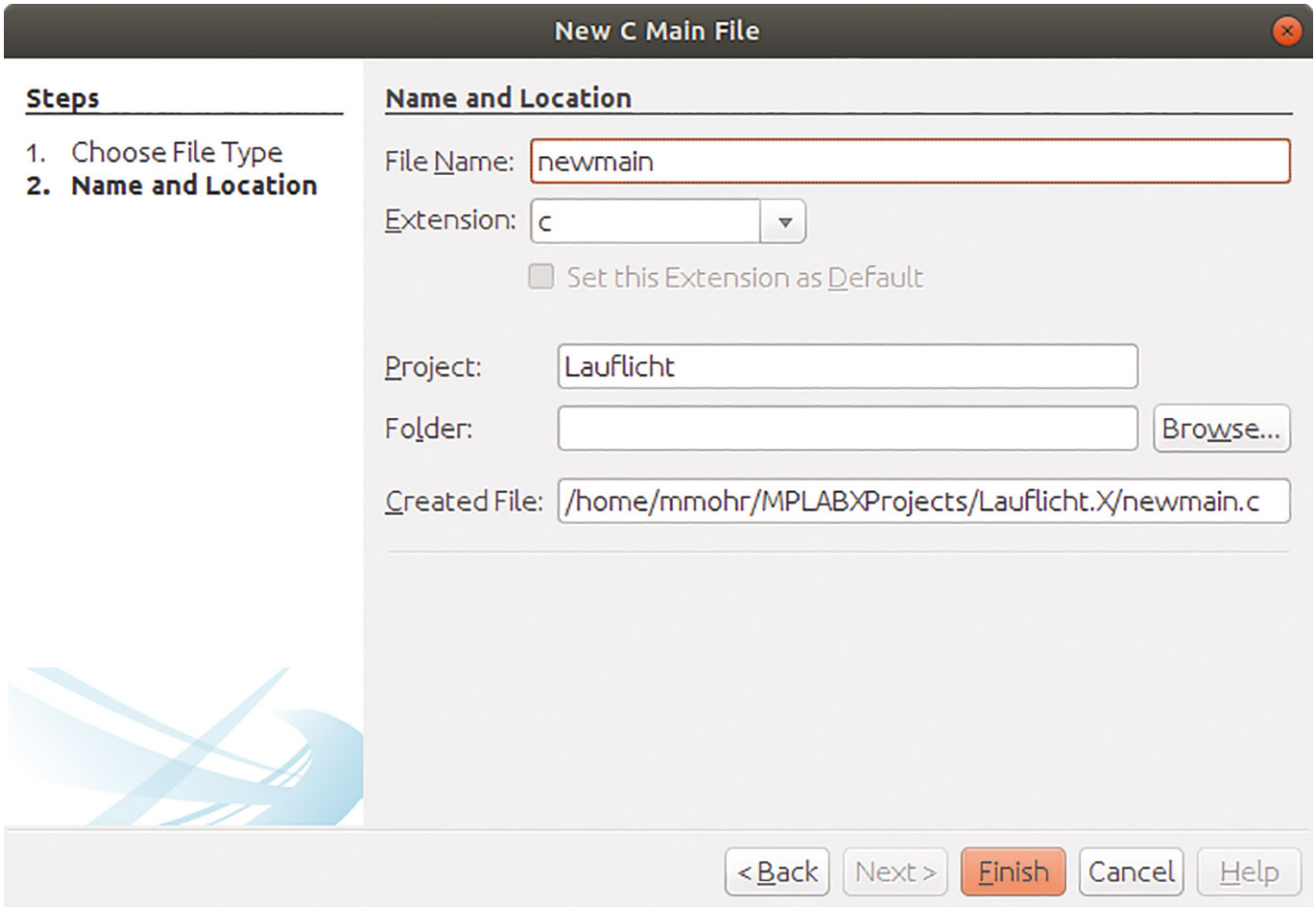


Nun müssen Sie den Compiler auswählen. Wenn bei der beschriebenen Installation alles geklappt hat, erscheint hier in der Liste der XC8-Compiler. Bitte wählen Sie ihn aus und klicken Next. Nun braucht unser Projekt noch einen Namen. Diesen können Sie im letzten Dialog des Wizards eingeben. Nun noch auf Finish klicken, und das Projekt ist fertig und wir können mit dem Programmieren beginnen.



Testprogramm

Normalerweise wird als erstes Beispiel bei der Mikrocontrollerprogrammierung immer eine LED zum Blinken gebracht. Das hier gezeigte Lauflicht hat nur wenig mehr Codezeilen, macht aber erheblich mehr her. Um das Programm einzugeben, klicken Sie nun mit der rechten Maustaste auf Source Files | New | C Main File. Es öffnet sich ein Dialog, mit dem Sie die Datei anlegen können (Abbildung unten). Nachdem Sie auf Finish geklickt haben, öffnet sich die neue Datei im Editor. Sie können nun den Beispielcode einfach übernehmen.



Sehen wir uns nun die einzelnen Codezeilen (Listing 1) etwas genauer an. Zuerst werden wie in C üblich die nötigen Bibliotheken importiert. Die zwei std*-Bibliotheken braucht man in der Regel immer, wenn man ein C-Programm schreibt. Interessant ist die xc.h. Mit ihr werden für die Programmierung wichtige Konstanten importiert. Ohne die xc.h könnten wir später im Programmcode nicht die sprechenden Klartextnamen der einzelnen Register oder Bits verwenden. Hier werden auch Initialisierungen, die für den fehlerfreien Betrieb des Mikrocontrollers wichtig sind, durchgeführt. Der nächste Befehl definiert die Frequenz des externen Quarzes. Das ist wichtig, weil sonst alle Timerfunktionen nicht korrekt arbeiten. In der nächsten Zeile beginnen einige allgemeine Konfigurationseinstellungen. Um es vorwegzunehmen, es gibt eine Unmenge dieser Schalter, die die Funktion des Mikrocontrollers beeinflussen. Hier werden nur einige gesetzt, die wichtig für das konkrete Programm sind.

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <xc.h>
#define _XTAL_FREQ 10000000

#pragma config FOSC = HS
#pragma config WDTE = OFF
#pragma config PWRTE = OFF
#pragma config CP = OFF

void main(void) {
  TRISB = 0;
```

```
  char out=1;
  char dir=1;
  while(1) {
    for ( int i=1;i<8;i++){
      PORTB = out;
      if (dir){
        out=out<<1;
      } else {
        out= out>>1;
      }
      __delay_ms(200);
    }//for
      dir= !dir;
  }//while
}// main
```

Mit #pragma config FOSC = HS definieren wir, dass sich an den Oszillatoreingängen des PIC16F84A ein High-Speed-Oszillator befindet. Wird dieser Parameter nicht gesetzt, startet der Controller nicht. Die möglichen Werte für diesen Parameter sind:

- LP (Low Power Crystal)
- XT (Crystal/Resonator)
- HS (High Speed Crystal/Resonator)
- RC (Resistor/Capacitor)

Eine detaillierte Beschreibung der Parameter finden Sie im Datenblatt, S. 22 f.

Der nächste Parameter schaltet den Watchdog-Timer aus: #pragma config WDTE = OFF. Falls er aktiv ist, wird er das Programm mitten in der Ausführung beenden und neu starten. Der Watchdog-Timer kann tatsächlich schwer zu findende Fehler verursachen. Wenn Sie eine Überwachung benötigen, müssen Sie Ihr Programm entsprechend schreiben. Der Timer wird zurückgesetzt, wenn die main-Funktion verlassen wird. Die main-Funktion wird dann wieder automatisch aufgerufen. Man spricht hierbei auch von einem zyklischen Ablauf des Programms. Eine genaue Erklärung zyklisch laufender Programme würde hier zu weit führen. Der Parameter #pragma config PWRTE = OFF aktiviert so etwas wie eine Einschaltverzögerung; sie soll den Controller beim Anschalten warten lassen, bis sich die Betriebsspannung stabilisiert hat. Wir benötigen diese Wartezeit hier nicht.

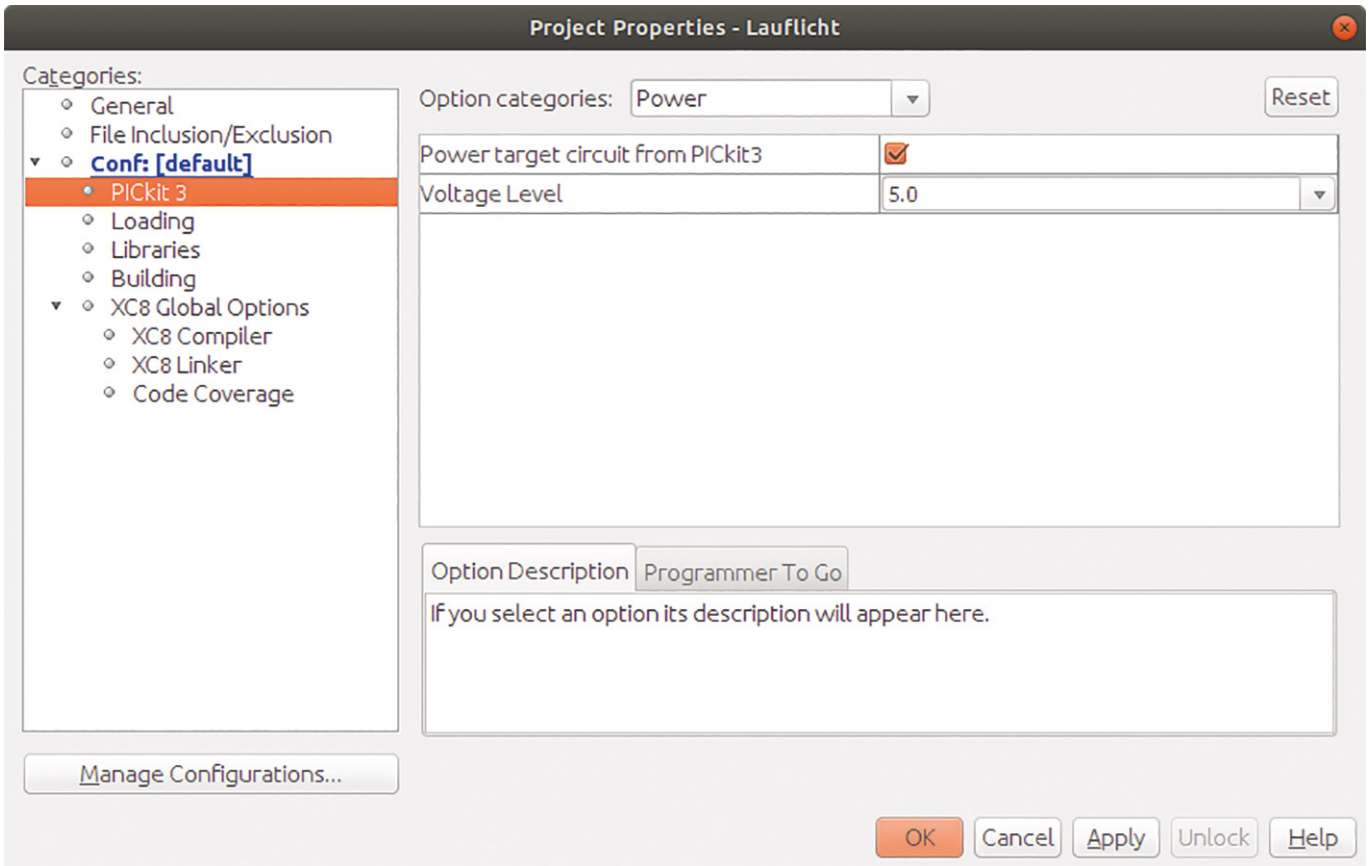
Der nächste Parameter kann einen Leseschutz für den internen Speicher aktivieren. Das bedeutet, dass man das Programm nicht mehr zurück aus dem Controller in die IDE lesen kann. Die Funktion ist dazu gedacht, das geistige Eigentum zu schützen. Wenn der Kopierschutz aktiviert ist, kann man den Baustein allerdings neu programmieren. Während der Entwicklung schadet es nicht, den Parameter zu deaktivieren: #pragma config CP = OFF.

Nun kommen wir endlich zur main()-Funktion. Mit TRISB = 0; konfigurieren wir den kompletten Port B als Output. Es gibt auch die Möglichkeit, die einzelnen Bits des Ports zu konfigurieren. Dazu würde man TRISB[0..7] verwenden. out ist die Bytevariable, in der der Wert ist, den wir später auf den Port schreiben. Die Variable dir wird als Boolean verwendet.

Wer schon einige Jahre mit moderneren Programmiersprachen gearbeitet hat, dem kommt der C-Typ char etwas eigentümlich vor. Man kann mit ihm eine Menge Sachen machen, die in moderneren Programmiersprachen einfach besser ausdefiniert sind, wie zum Beispiel Boolean.

Die while-Schleife in der nächsten Zeile ist eine Endlosschleife. Sie sorgt dafür, dass unser Programm ewig läuft. Die nun folgende for-Schleife schiebt das einzelne Bit in der Variable out im ersten Durchlauf von links nach rechts (weil die Variable dir 1 ist). Wenn die for-Schleife beendet wird, wird die Variable dir (für Direction) negiert, um danach direkt wieder zu beginnen. Mit dem Unterschied, dass das Bit diesmal von rechts nach links geschoben wird. Nun ist der Zeitpunkt gekommen, an dem die Funktion immer wieder von vorne startet. Die einzige Möglichkeit, sie zu beenden, ist entweder eine Neuprogrammierung des Mikrocontrollers oder einfach ein Abschalten der Spannung.

Bevor wir mit dem Programmieren beginnen können, müssen wir noch eine kleine Einstellung in den Eigenschaften des Programmieradapters vornehmen. Klicken Sie mit der rechten Maustaste auf das Projekt (hier: Lauflicht) und wählen Sie die Properties aus. Nun öffnet sich ein Dialog, klicken Sie hier im Baum auf der rechten Seite auf den PICkit3. Auf der linken Seite wählen Sie nun Power aus. Bitte machen Sie den Haken in das Feld Power target circuit from PICkit3, wie es auf nachstehender Abbildung zu sehen ist. Diese Option aktiviert die Spannungsversorgung aus dem PICkit3. Wenn Sie mit einem zusätzlichen externen Netzteil arbeiten, müssen Sie die Haken wieder entfernen. Klicken Sie nun Apply und dann Ok.



Um das Programm zu bauen und in den Controller zu laden, klicken Sie mit der rechten Maustaste auf das Projekt und wählen Run aus. Die IDE beginnt nun automatisch das Projekt zu bauen. Ist der Bau erfolgreich abgeschlossen, wird das Programm automatisch in den PIC16F84A geladen und dort gestartet. Alternativ zu Run können Sie auf den kaum zu übersehenden Button mit dem grünen Pfeil nach rechts klicken.

Falls aus einem unerfindlichen Grund einmal das Schreiben des Programms nicht möglich ist, hilft es sehr oft, den Programmieradapter kurz vom USB zu trennen und neu zu verbinden. Als zweiten Trick können Sie die IDE noch einmal neu starten. Manchmal werden die Parameteränderungen am Programmieradapter erst nach dem Neustart der IDE übernommen.

Fazit

Die Mikrocontroller der PIC-Familie lassen sich mit der MPLAB X IDE superleicht programmieren. Die Controller-Familie ist groß genug, um für jeden nur erdenklichen Anwendungsfall den richtigen Baustein zu finden. Falls Sie sich mit der Assemblersprache etwas beschäftigen wollen, ist dieser Controller der ideale Einstieg. Die 35 möglichen Kommandos, die er versteht, hat man schnell gelernt. Alternativ kann man ihn auch in C programmieren. Die Teile, die man für einen Einstieg benötigt, kosten unter 20 Euro. Alles in allem eine schöne Beschäftigung für einen verregneten Sonntagnachmittag.

PIC,, Programmierung,, Einstieg,, Erste, Schritte