Zielstellung

Dieser Schnelleinstieg demonstriert das Arbeiten mit dem Klassendiagramm in SiSy AVR mit der Bibliothek für das myAVR Board MK3.
Es werden alle Schritte gezeigt, die zur Erstellung eines Programms für einen Mikrocontroller notwendig sind. Zur Verwendung kommt die Projektvorlage „myAVR Board MK3“, in der bereits wichtige Klassen vorhanden sind.
Außerdem wird eine allgemeine Rahmenanwendung für Mikrocontroller vorgestellt.
Weitergehende Beschreibungen zum Arbeiten mit SiSy finden Sie im SiSy Benutzerhandbuch.

Voraussetzungen

Für die Bearbeitung der Aufgaben benötigen Sie folgende Software und Hardware:

Software

- SiSy ab Version 3
- SiSy Microcontroller++ oder
  SiSy AVR oder
  SiSy Ausgabe Developer, Professional bzw. BS
  mit dem integriertem ***Add-On AVR***
- Die myAVR Bibliothek „Projektvorlage für das myAVR Board MK3“

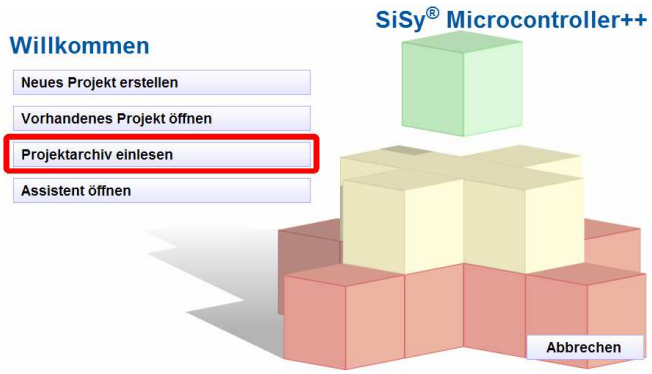
Hardware

- myAVR Board MK3

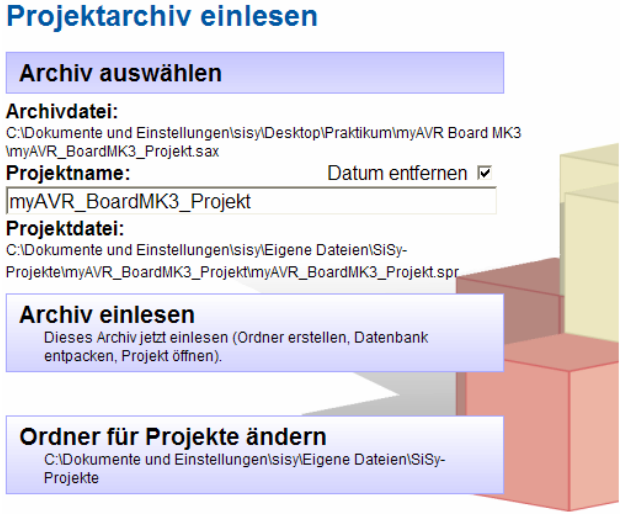
Verbinden Sie die Ports D0 bis D2 mit der LED-Ampel. Dies entspricht auch dem Auslieferungszustand des myAVR Board MK3.

1. Ein neues Projekt anlegen

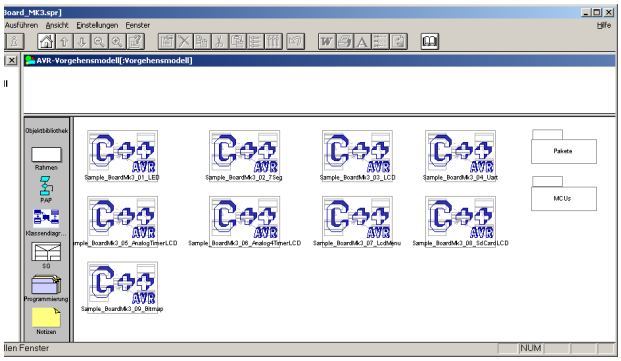
Starten Sie SiSy und wählen Sie im Startbildschirm *Projektarchiv einlesen*.



Im folgenden Dialogfenster laden Sie die Archivdatei *myAVR_BoardMK3_Projekt.sax*. Speichern Sie das Projekt auf jeden Fall in einem eigenen Verzeichnis.

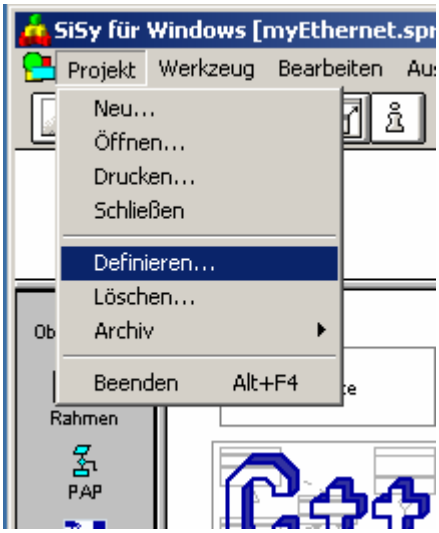


Das Projekt wird in der Ansicht *Vorgehensmodell* geöffnet. Dieses Diagramm bildet den Ausgangspunkt für die weitere Bearbeitung.

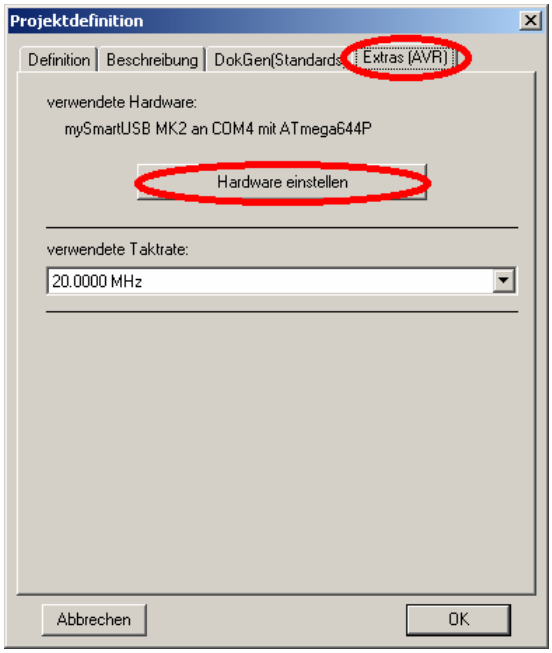


2. Hardware einstellen

Als erstes sollte das soeben geladene Projekt an Ihre aktuelle Hardwarekonfiguration angepasst werden.
Öffnen Sie dazu den entsprechenden Dialog über die Menüpunkte *Projekt->Definieren...*



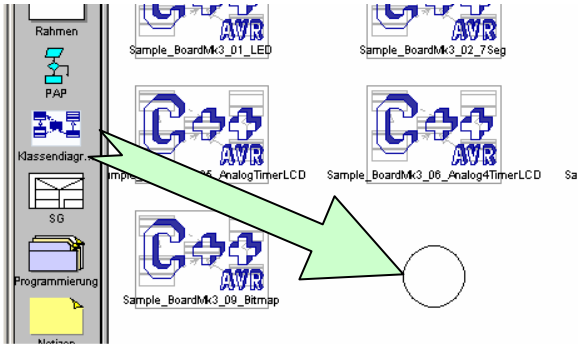
Wählen Sie die Registerkarte *Extras (AVR)* und dort die Schaltfläche *Hardware einstellen*.



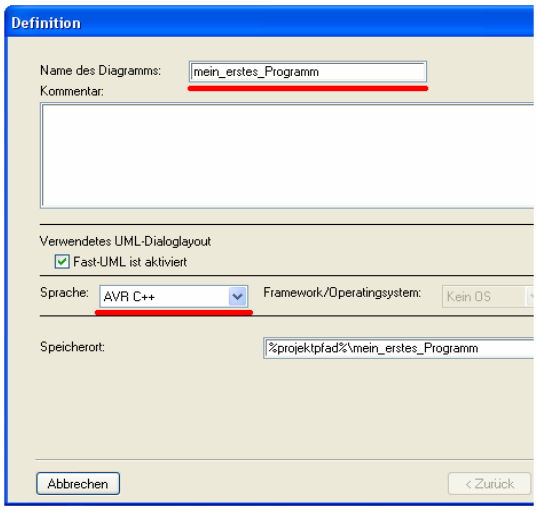
Es öffnet sich das Programm zum Einstellen der Programmierverbindung. Stellen Sie Ihre vorhandene Programmierhardware und den Controller ein
(im Bild: **mySmartUSB MK3** und **Atmega2560**).

3. Neues Programm erstellen

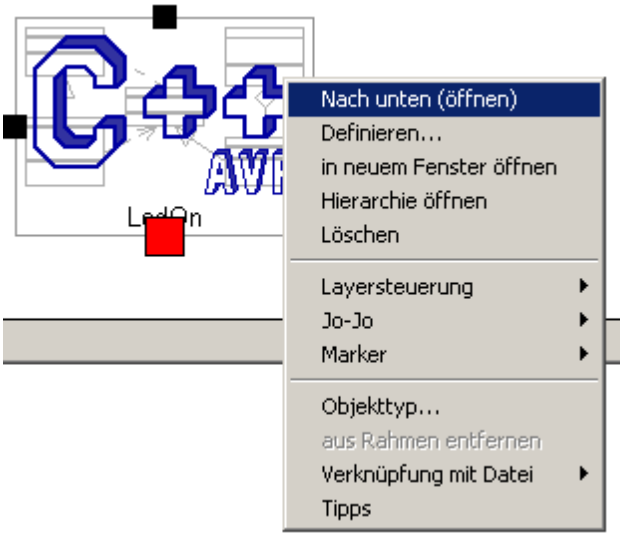
Legen Sie ein neues Programm an. Ziehen Sie dazu ein Klassendiagramm aus der Objektbibliothek



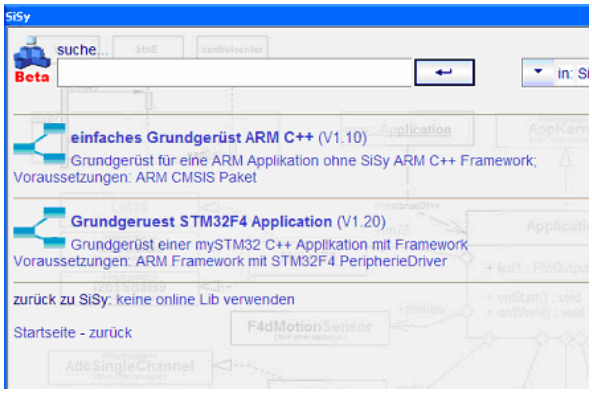
Geben Sie dem Programm einen Namen z.B. mein_erstes_Programm. Achten Sie darauf, das als Sprache AVR C++ eingestellt ist. Ein Unterordner für Ihr Programm wird automatisch angelegt.



Wechseln Sie in Ihr neues Diagramm, indem Sie das Diagrammobjekt mit der rechten Maustaste anklicken und im erscheinenden Menü den Punkt *Nach unten (öffnen)* auswählen.



Im folgenden Dialog kann eine Programmvorlage gewählt werden.

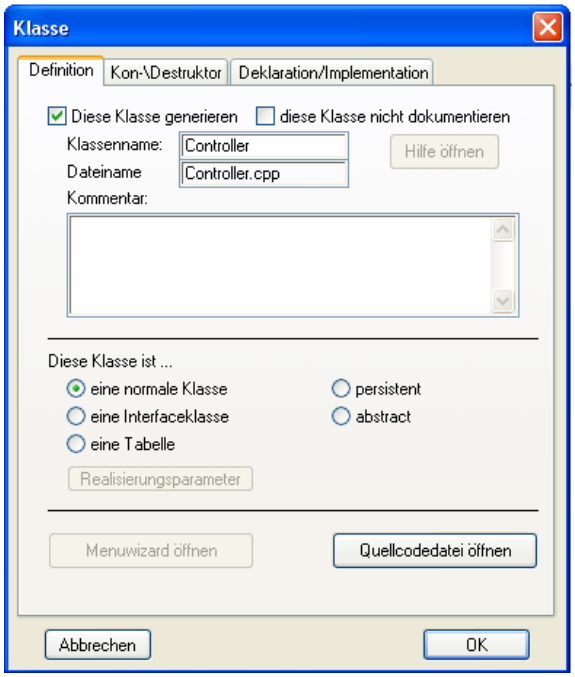


In diesem Beispiel sollen Sie jedoch alle Elemente selbst erstellen. Schließen sie das Fenster. Sie sollten jetzt das leere Klassendiagramm sehen. Für spätere Projekte können Sie natürlich das angebotene Grundgerüst verwenden.

4. Anwendung Grundgerüst

Als erstes wird ein Anwendungs-Grundgerüst erstellt. Dieses kann für jedes Mikrocontroller-Programm verwendet werden.

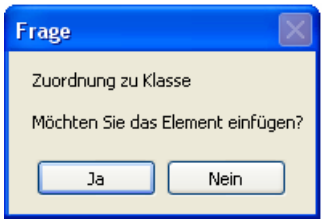
Ziehen Sie eine Klasse aus der Objektbibliothek in Ihr Diagramm. Diese Klasse repräsentiert das Programm auf dem Mikrocontroller.
Es öffnet sich ein Dialog, in dem Sie den Namen der Klasse festlegen. Setzen Sie außerdem den Haken in der Checkbox *Diese Klasse generieren*.



Der grundsätzliche Programmablauf besteht aus

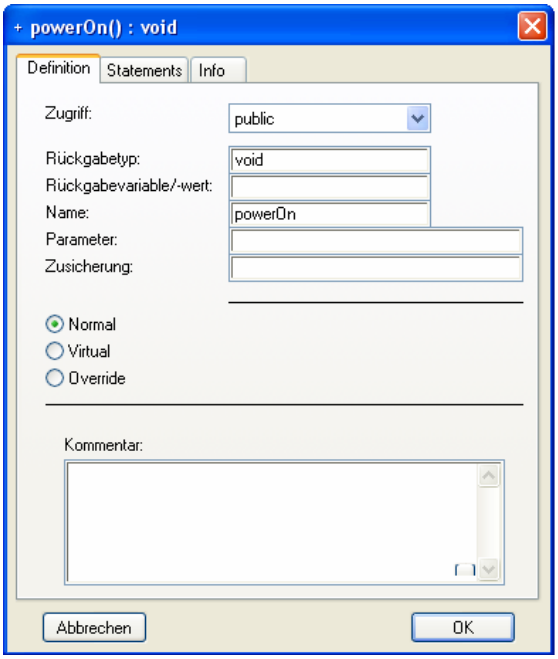
1. Initialisierungen
2. Programmdurchführung (Main-Loop)

Ziehen Sie dazu nacheinander 2 Operationen aus der Objektbibliothek auf die Klasse *Controller*. Beantworten Sie die Fragestellung der MessageBox mit *Ja*.

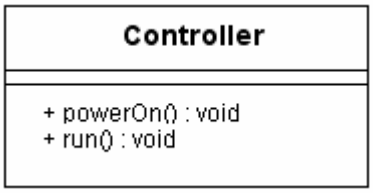


Sollte die MessageBox nicht erscheinen, haben Sie die Klasse nicht getroffen. Sie können die Operation später durch Verschieben auf die Klasse hinzufügen.

Es öffnet sich der Dialog für die neue Operation. Legen Sie die Funktionsnamen *powerOn* und *run* fest.

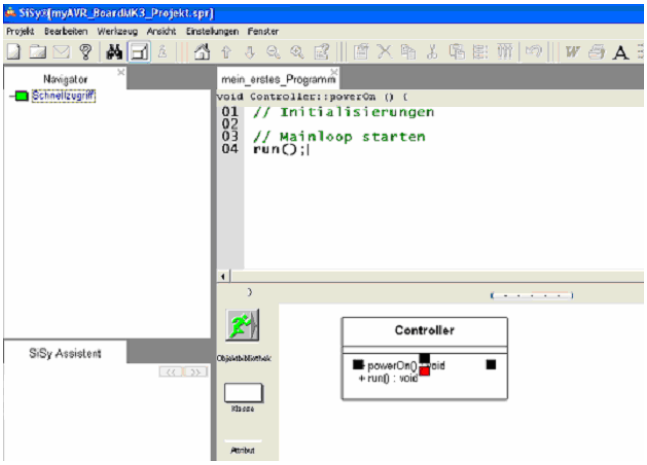


Die Klasse sollte jetzt folgendes Aussehen haben

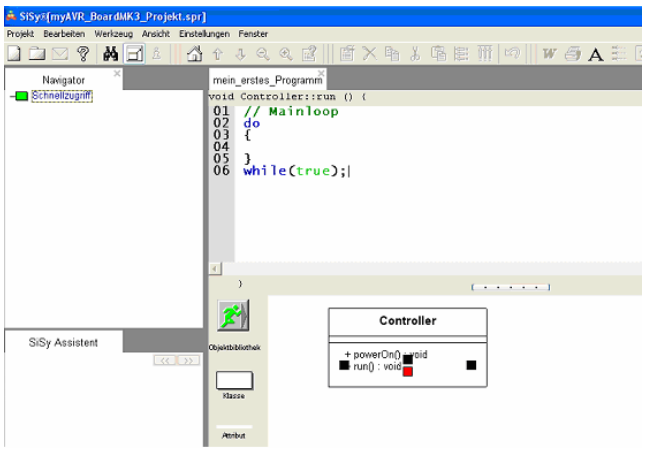


Wenn Sie eine der Operationen markieren, wird im oberen Teil des Diagramms der Quelltext dieser Funktion dargestellt. Mit der Taste F4 kann die Größe des Quelltextfensters umgeschaltet werden.
Der Programmablauf soll Folgender sein:

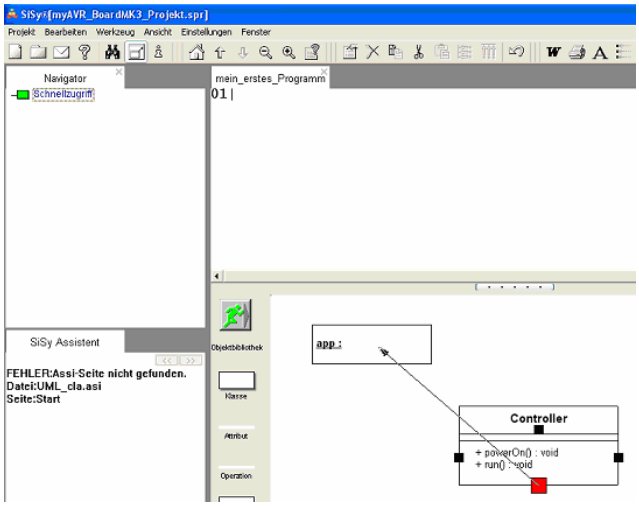
- Beim Programmstart (Reset oder Power On) wird die Operation *powerOn* ausgeführt. Hier werden alle Initialisierungen durchgeführt.



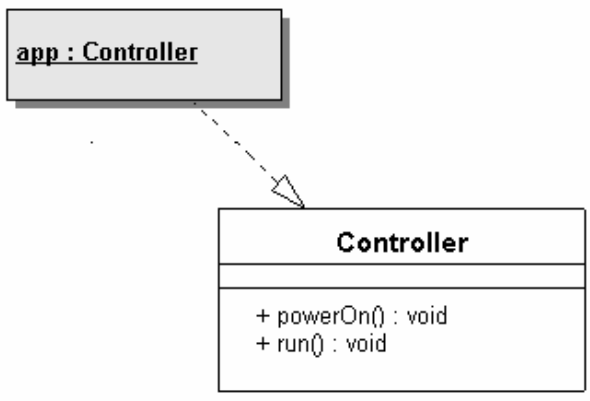
- Danach wird die Mainloop gestartet.



Um das Grundgerüst fertigzustellen, wird noch eine Instanz der Controller-Klasse benötigt. Ziehen Sie dazu ein Objekt aus der Objektbibliothek in Ihr Diagramm und geben Sie diesem Objekt den Namen *app*. Markieren Sie die Klasse *Controller* (Achtung: nicht eine der Operationen, sondern den Klassenrahmen markieren) und ziehen Sie eine Verbindung von dem unteren (roten) Marker der Klasse zum gerade angelegten Objekt *app*.



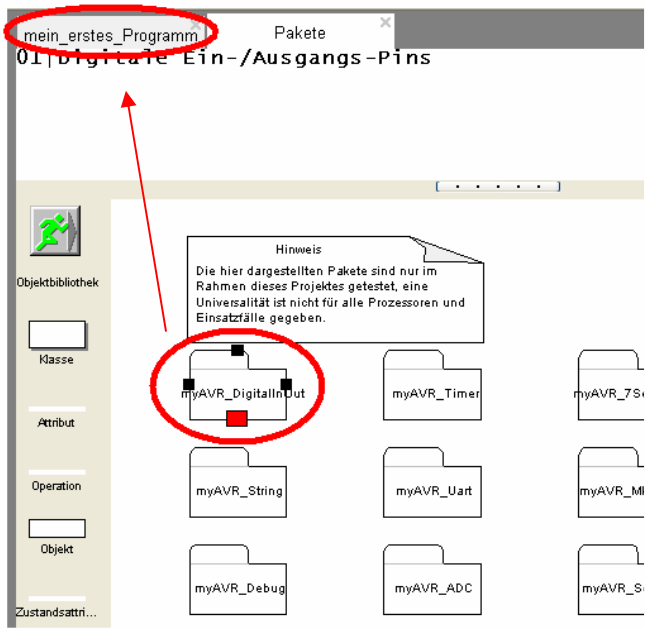
Bestätigen Sie den Dialog und der Klassenname sollte im Objekt eingetragen werden.
Das fertige Klassendiagramm des Grundgerüstes hat jetzt folgendes Aussehen:



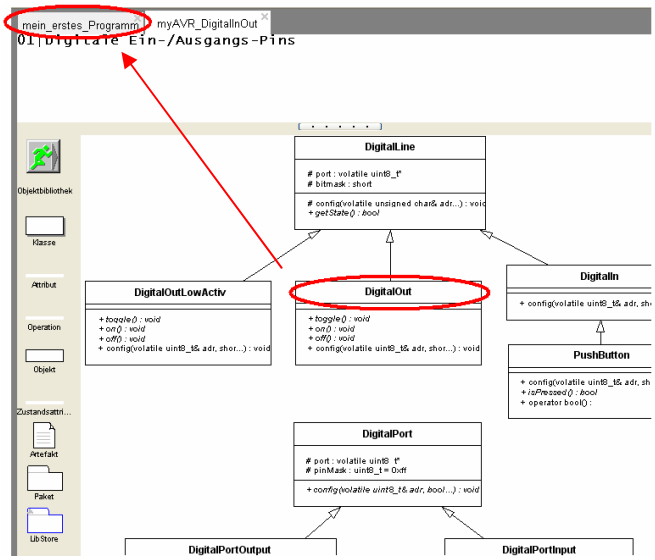
5. Bibliothek verwenden – LED einschalten

Als nächstes soll eine der Klassen aus den Bibliotheks-Paketen verwendet werden. Hier wird *DigitalOut* benutzt, um die LEDs zu schalten.
Öffnen Sie dazu ein neues Diagrammfenster.

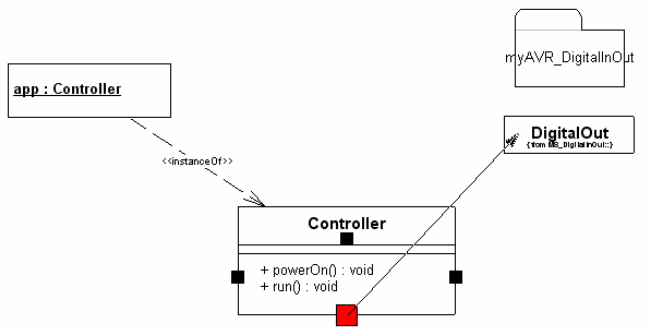
Wechseln Sie im neuen Fenster auf Pakete nach unten.
Ziehen Sie das Paket *myAVR_DigitalInOut* in „mein_erstes_Programm“.



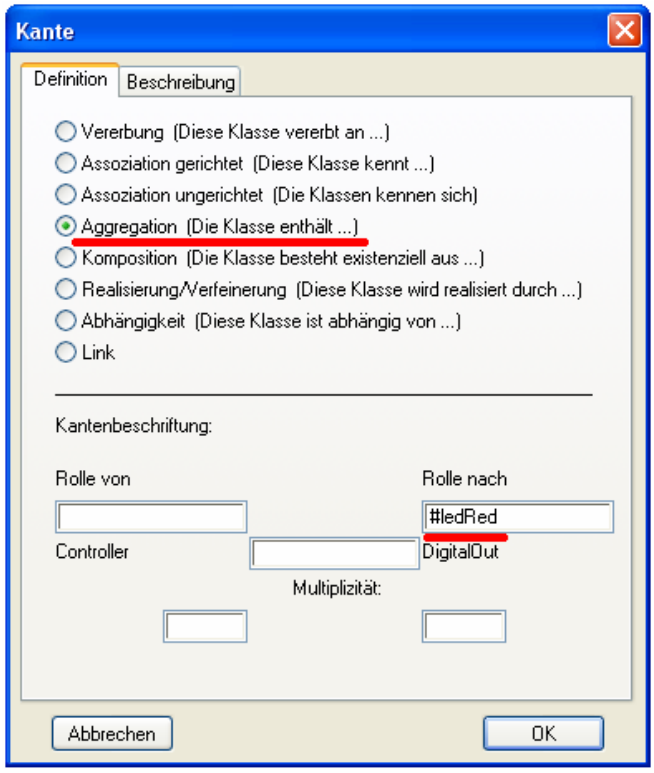
Wechseln Sie danach in das Paket (*Nach unten (öffnen)*) und ziehen Sie auch die Klasse *DigitalOut* in Ihr Programm.



Erstellen Sie eine Verbindung von der Klasse *Controller* zu *DigitalOut.*



Wählen Sie den Verbindungstyp *Aggregation (Die Klasse enthält ...)* und tragen Sie im Feld *Rolle nach* ein: *#ledRed* . Damit fügen Sie Ihrer Anwendung eine Instanz der Klasse *DigitalOut* mit diesem Variablennamen hinzu.



Die Klasseninstanz muss jetzt noch initialisiert werden. Dies erfolgt in der *powerOn*-Methode. In diesem Beispiel soll die rote LED mit Port D, Bit 0 verbunden sein.

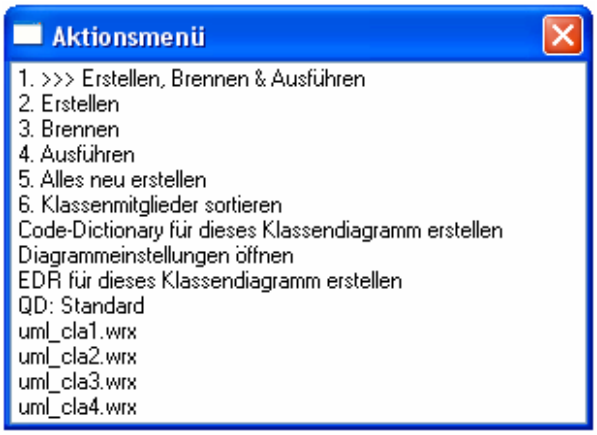
```
void Controller::powerOn () {
01  // Diese Operation als Start-Operation für main angeben
02  // hier Initialisierungen durchführen
03  ledRed.config(PORTD, BIT0);
04
05  // Mainloop starten
06  run();
```

In der Mainloop, der *run*-Methode, kann die LED jetzt geschaltet werden.

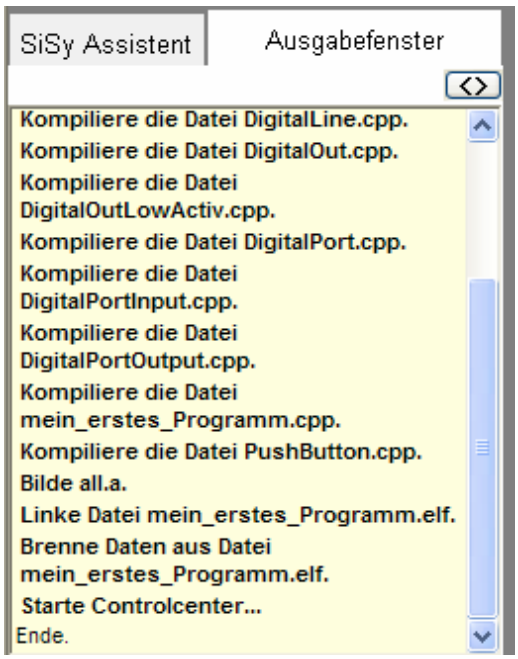
```
void Controller::run () {
01  // Mainloop
02  do
03  {
04      ledRed.toggle();
05      waitMs(300);
06  }
07  while(true);
```

6. Erstellen und Brennen

Das Programm kann jetzt erstellt und gebrannt werden. Klicken Sie dazu auf die Schaltfläche für das Aktionsmenü in der Objektbibliothek.



Wählen Sie Punkt *1.*
>>> Erstellen, Brennen & Ausführen.
Beim ersten Erstellvorgang öffnet sich ein Dialog, in dem die Startfunktion festgelegt werden soll. Wählen Sie die Operation *powerOn*.

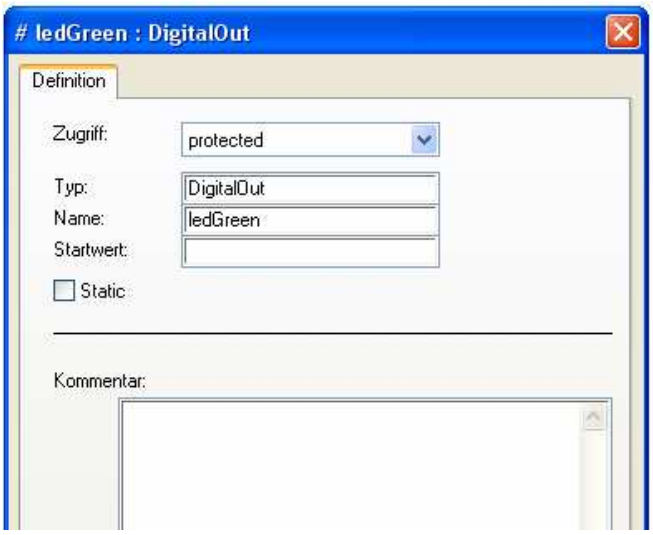


Beim Erstellen werden Dateien verwendet, die in diesem Diagramm nicht angelegt wurden (z.B. *PushButton.cpp*). Diese stammen aus dem Paket der Klassenbibliothek. Die Auswahl, welche Klassen vom Programm tatsächlich verwendet werden, erfolgt erst beim Linken. Das heißt, die zusätzlichen Klassen sind im fertigen Programm auf dem Mikrocontroller nicht enthalten.

Nach erfolgreichem Übersetzen und Brennen blinkt jetzt die rote LED.

7. Zweite LED

Für die grüne LED soll noch eine andere Methode zur Verwendung der Bibliothek demonstriert werden.
Ziehen Sie dazu aus der Objektbibliothek ein Attribut auf die Klasse *Controller*. Es erscheint wieder die Frage nach dem Einfügen in die Klasse und danach ein Dialog.



Geben Sie als Typ die Klasse aus der Bibliothek (hier *DigitalOut*) an und legen Sie einen Namen fest.
Das neue Attribut kann genauso verwendet werden wie das bereits vorhandene.

Die Quelltexte der Operationen *powerOn* und *run* können jetzt ergänzt werden. Die grüne LED ist an Port D, Bit 2 angeschlossen.

```
void Controller::powerOn () {
43  // Diese Operation als Start-Operation
44  // hier Initialisierung durchführen
45  ledRed.config(PORTD, BIT0);
46  ledGreen.config(PORTD, BIT2);
47
48  // Mainloop starten
49  run();
```

```
void Controller::run () {
59  do
60  {
61      ledRed.toggle();
62      waitMs(200);
63      ledGreen.toggel();
64      waitMs(100);
65  }
66  while(true);
```

Nach erneutem Erstellen und Brennen blinken jetzt sowohl die rote als auch die grüne LED.