Detaillierte Erklärung

```
#include <Wire.h>
```

Mittels # werden Prä-Prozessor-Anweisungen gekennzeichnet. Diese Zeilen sind keine Befehle, sondern steuern den Compiler und seine Hilfsprogramme. Es darf hier kein Semikolon stehen und Kommentare in der gleichen Zeile führen zu kryptischen Fehlermeldungen...
#include <Wire.h> bedeutet, dass der Compiler Zugriff haben soll auf die Bibliothek Wire.h, die I2C-Bibliothek. Die spitzen Klammern sagen, dass die Library in Systemverzeichnissen von Arxduino liegt.

```
// Anm. 1
#define DAC_ADR 0x48
#define ENABLE_OUTPUT B01000000
#define ledPin 2
```

Hier werden symbolische Konstanten definiert. Warum macht man das ? Es verbessert die Lesbarkeit und ersetzt bei Änderungen der Hardware bei großen Programmen eine fehlerträchtige Suchen-und-Ersetzen-Aktion. Was bedeutet 48 hex mitten im Programmtext ? Es könnte eine beliebige Bedeutung haben, DAC_ADR ist immer die Adresse des DA-Wandlers. Die Zeile in diesem Absatz werden vor der Compilation von der Compilerfunktion Prä-Prozessor mittels Textersatz in den programmtext eingesetzt.

```
//Anm.3
byte stat = 0;
byte n = 0;
```

Dieser Absatz reserviert Speicherplätze für Variablen. Variablen beinhalten Werte, die im Programmablauf verändert werden. Vor dem Variablenname steht der Typ byte. Der Variablentyp sagt aus wie breit der reservierte Speicherplatz sein soll. Das Gleichzeichen bedeutet, dass die Variable auch gleich mit einem Initialsierungswert gefüllt werden soll. Es ist also ein Befehl (bevor das Programm beginnt!) und hat daher ein Semikolon am Ende. Die Deklaration der Variablen an dieser Stelle bedeutet, dass sioe im gesamten Programmbereich und allen Unterprogrammen zugegriffen werden kann (global).

```
//Anm.4
void setup()
{
    ……..
}
```

Es wird das Unterprogramm (C: die Funktion) setup aufgerufen. Void bedeutet, dass diese Funktion keinen Wert zurückliefert, wenn sie beendet wird. () bedeutet, dass sie keinen Wert beim Start übergeben bekommt. In setup() werden alle Variablen für den Start vorbereiotet und Hardware initialisiert. Libraryfunktionen werden ebenfalls initlisiert, damit sie arbeitfähig sind.
Diese Funktion wird in „normalem“ C durch den Anwender im Programm main() mit gleicher Bedeutung als erstes gestartet. In Arduino wird das Programm main() automatisch mit generiert und setup() entsprechend eingefügt.

```
void setup()
{
    //Anm.5
    // Verwendete Bibliotheken
    Wire.begin();
    Serial.begin(9600);

    Serial.print("Start ");
    // Ausgang für LED deklarieren
    pinMode(ledPin, OUTPUT);
}
```

Wire.begin() ist ein Aufruf einer Funktion der Wire-Library, der eine Initialisierung aller Variablen und eine Initialisierung der I2C-Hardware durchführt.
Serial.begin() ist ein Aufruf einer Fuinktion der Serial-Library, der eine Initialisierung der Variablen des Software-treiberfunktion und der Hardware der seriellen Schnittstelle mit der Baudrate von 9600 Bit/s einrichtet.
Serial.print(„Start“) bedeutet einen Ausdruck des Textes Start als Kennzeichnung des Programmstarts auf dem seriellen Monitor.
pinMode(ledPin, OUTPUT) initialisiert den Prozessorpin D2 als Output. Die Konstante OUTPUT wird dabei in der Serial-Library definikert und ist in der Beschreibeung der Serial-Library zu finden.
Damit ist alles vorbereitet für den eigentlichen Programmstart

```
void loop()
{
   ..............
}
```

Mikrocontroller-Systeme haben häufig kein Betriebssystem. Wenn das System gestartet ist läuft das Programm (oder ein Teil davon) im Dauerbetrieb bis die Versorgungsspannung abgeschaltet wird. Das Unterprogramm loop() wird in „normalen“ C-Systemen im Hauptprogramm main.c als Endlosschleife eingebaut. Die Entwicklungsumgebung des Arduino setzt vor der Compilation den Inhalt von loop() entsprechend ein.

```
void loop()
{
   //Anm.6
   // Daten werden staendig erhoeht und ausgegeben (Rampe)

   //Anm.7
   Wire.beginTransmission(DAC_ADR);     // Telegramm eröffnen zu Slave 48h
   Wire.write(ENABLE_OUTPUT);           // Control-Register in Puffer fuellen
   Wire.write(n);                       // Daten-Byte in Puffer füllen
   stat = stat | Wire.endTransmission();     // Telegramm absenden und
                                        // Fehlermeldungen kumulieren
   Serial.print(n);
   Serial.println(" ");

   n++;              // Datenwert erhöhen für nächste Ausgabe
   delay(10);        // Pause

   ……..
}
```

Hier wird die Hauptaufgabe des Programms durchgeführt. Das Telegramm wird spezifisch für den Baustein ausgeführt. Es werden laut Datenblatt bei jedem Zugriff 2 Bytes übertragen: ein Control-Register und der auszugebende Datenwert.

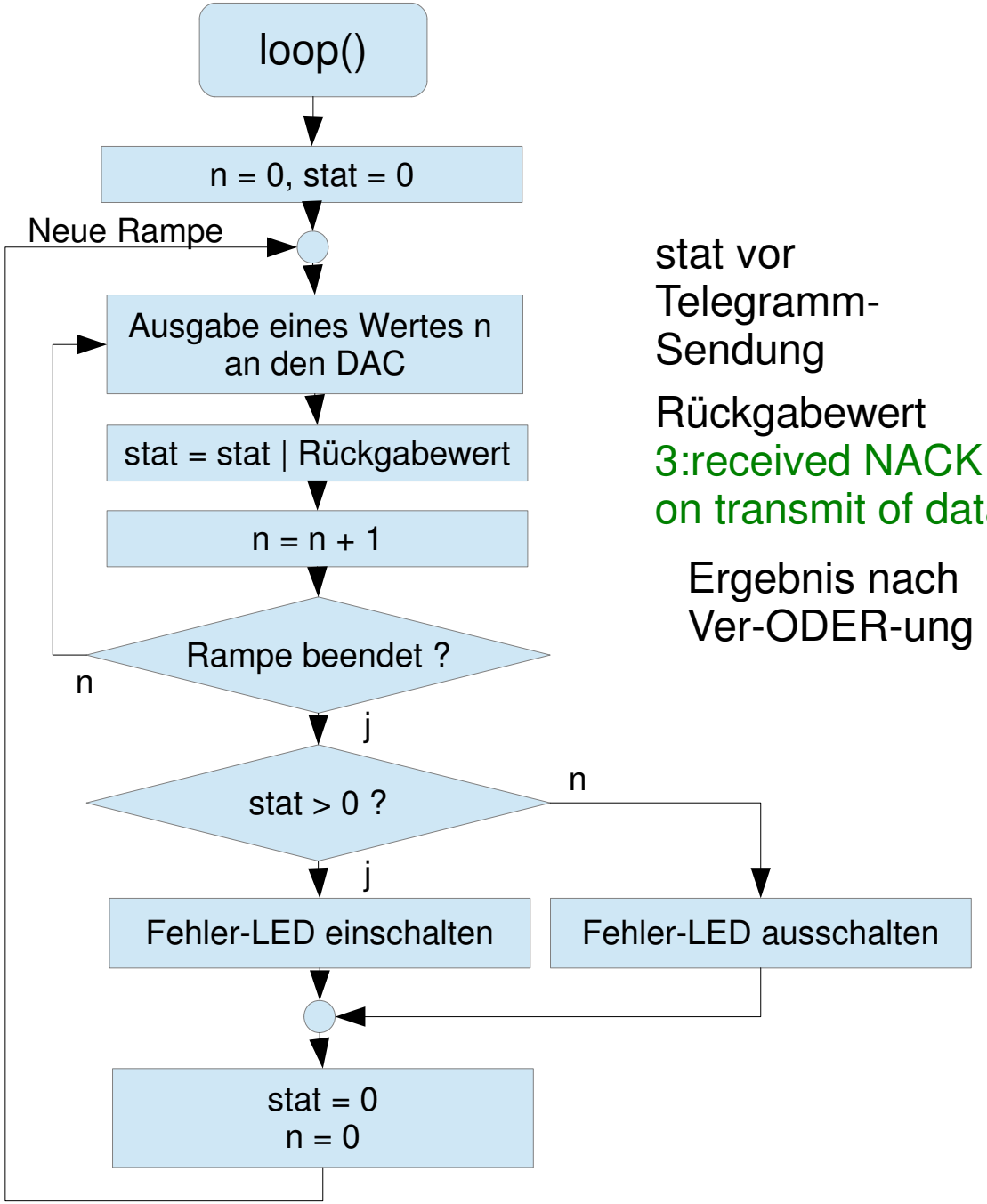
```
//Anm.7
Wire.beginTransmission(DAC_ADR);     // Telegramm eröffnen zu Slave 48h
Wire.write(ENABLE_OUTPUT);           // Control-Register in Puffer fuellen
Wire.write(n);                       // Daten-Byte in Puffer füllen
Wire.endTransmission();              // Telegramm absenden
```

Wire.beginTransmission(DAC_ADR); startet das Anlegen eines Datensatzes für ein Telegramm an den Slave mit der Adresse DAC_ADR (48hex).Wire.write(ENABLE_OUTPUT); schreibt die Konstante ENABLE_OUTPUT (B01000000)in den Datensatz als Byte 1. Wire.write(n);schreibt den Wert der Variable n (Zahl zwischen 0 und 255) in das zweite Byte. Wire.endTransmission(); sendet nun das Telegramm ab und die Funktion setzt dabei das Telegramm aus dem Datensatz-Einträgen zusammen.
Wenn man in der Beschreibung der Wire-Library nachschlägt, dann kann byte Wire.endTransmission() bei Beendigung der Funktion ein Byte zurücksenden, dass das Ergebnis der Operation zeigt:

0:success
1:data too long to fit in transmit buffer
2:received NACK on transmit of address
3:received NACK on transmit of data
4:other error

Der Aufruf kann dann z.B.
Wire.endTransmission(); lauten. In der Variablen stat finden sich dann die oben beschriebenen Werte und können ausgewertet werden.

Im Programm steht jedoch
stat = stat | Wire.endTransmission();
Der Fehlerstatus soll später nuir einmal pro Rampendurchlauf kontrolliert werden, ob er grösser als Null ist. Ein einfacher Weg ist die bitweise Veroderung aller Werte in stat hineine. Jeder Wert wird bitweise auf den vorhandenen Wert in stat hinzuge-Odert. Ict also in einer der Rückgabewerte eine 1 enthalten, dann muss dieser Wert > 0 sein.

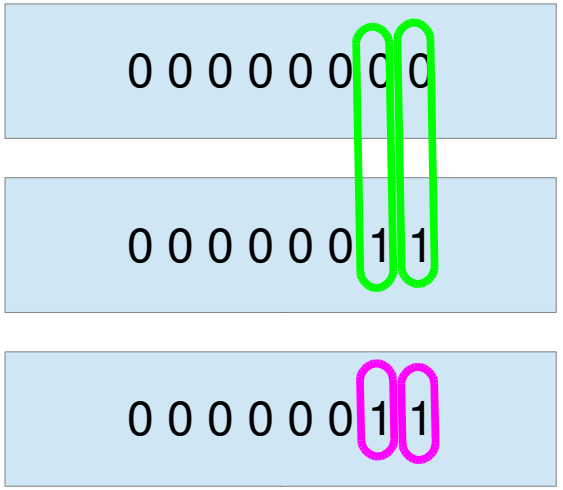


stat vor Telegramm-Sendung

Rückgabewert
3:received NACK
on transmit of data

Ergebnis nach Ver-ODER-ung

Bitweise Ver-ODER-ung



ODER-Funktion

Bit1	Bit2	Erg.
0	0	0
0	1	1
1	0	1
1	1	1

```
void loop()
{
    ……..
    n++;                // Datenwert erhöhen für nächste Ausgabe
    delay(10);          // Pause 10ms


    if(n == 255)
    {
      ……..
      // Fehlermerker loeschen
      stat = 0;
      n = 0;
      Delay(1000);   // Wartezeit 1s
    }
}
```

Nach der Ausgabe des Wertes der Variablen n wird der Wert mit n++ um den Betrag erhöht und vor der nächsten Ausgaberunde 10ms gewartet. Dann erfolgt mit if(n == 255) die Abfrage on n gernau dem Wert 255 entspricht. Dieser Wert ist der grösste Wer, der in einem Byte dargestellt werden kann und unser höchster Ausgabewert. Damit ist erkannt, die Rampe ist oben angekommen und soll nun ab Null wieder starten. Dazu wird der Fehlermerker stat gelöscht und der Ausgabewert n auf Null gesetzt und 1 s gewartet. Danach geht es in der Endlosshleife wieder mit der nächsten Rampe weiter.

```
if(n == 255)
{
  //LED einschalten, wenn im letzten Durchlauf ein Fehler im Fehlermerker steht
  if(stat != 0)
  {
    digitalWrite(ledPin,1);
  }
  else
  {
    // Kein Fehler, dann LED loeschen
    digitalWrite(ledPin,0);
  }
  // Fehlermerker loeschen
  stat = 0;
  n = 0;
  delay(1000);
}
```

Bevor die nächste Rampe gestartet wird, wird mit if(stat != 0) der Wert stat untersucht, ob er ungleich Null ist. Wenn das so ist, dann ist in einem der Durchläufe der letzten Rampe ein Fehler aufgetreten und die LED wird mit digitalWrite(ledPin,1); eingeschaltet. Wenn jedoch stst gleich Null ist, dann wird die LED mit digitalWrite(ledPin,0); gelöscht.