Technische Universität Chemnitz

Fakultät für Elektrotechnik und Informationstechnik

Professur Mikrotechnologie

Studienarbeit

Entwurf eines Prozesssteuermoduls zur Atomlagenabscheidung

Bearbeiter :	Michael Böhme
Betreuer der Arbeit :	Dr.-Ing. S. E. Schulz Dipl.-Ing. T. Wächtler
Verantwortlicher Hochschullehrer :	Prof. Dr.-Ing. habil. T. Geßner
Tag der Ausgabe :	1. Dezember 2004
Tag der Abgabe :	15. September 2005

Inhaltsverzeichnis

Technische Universität Chemnitz

Fakultät für Elektrotechnik und Informationstechnik

Professur Mikrotechnologie

Aufgabenstellung

für eine

Studienarbeit

Name, Vorname:	B Ö H M E , Michael
geb. am:	26. August 1981
Studiengang:	Informations- und Kommunikationstechnik
Matrikel-Nummer:	31105

Thema: Entwurf eines Prozesssteuermoduls zur Atomlagenabscheidung

(ausführliche Aufgabenstellung siehe Rückseite)

Die wissenschaftliche Arbeit ist als Einzelarbeit studienbegleitend anzufertigen.

Betreuer der Arbeit:	Dr.-Ing. S. E. Schulz Dipl.-Ing. T. Wächtler
Tag der Ausgabe:	1. Dezember 2004
Abgabetermin:	1. Mai 2005
Tag der Abgabe:	

Dr.-Ing. S. E. Schulz
Betreuer

Prof. Dr.-Ing. habil. T. Geßner
Verantwortlicher Hochschullehrer

Ausführliche Aufgabenstellung:

Für die chemische Gasphasenabscheideanlage „Varian Gartek“ soll ein Prozesssteuermodul entworfen werden, das in der Lage ist, Ventile zur Gasfluss- und Vakuumkontrolle zu schalten. Die zu entwerfende Schaltung soll durch ein ebenfalls zu entwickelndes grafisches PC-Interface gesteuert werden, mit dem der Benutzer die Möglichkeit hat, die Schaltreihenfolge und damit den Abscheideprozess flexibel zu beeinflussen. Das Ziel sind zyklische Prozesse für die Atomlagenabscheidung (Atomic Layer Deposition, ALD), bei denen eine beliebige Ventil-Schaltsequenz periodisch wiederholt wird.

Teil der zu erledigenden Arbeiten ist die Wiederinbetriebnahme der Prozesssteuereinheit „Tymgard Process Controller“, die an der Abscheideanlage vorhanden ist. Diese ist jedoch nur sehr eingeschränkt für ALD-Prozesse nutzbar und bietet nicht die für zyklische Prozesse erforderliche Flexibilität. Deswegen ist eine speziell konzipierte Steuereinheit nötig. Um jedoch kurzfristig bereits erste Atomlagenabscheideprozesse durchführen zu können, soll die Tymgard-Einheit durch eine entsprechende äußere Beschaltung modifiziert werden.

Arbeitsaufgaben:

- Wiederinbetriebnahme der Prozesssteuereinheit „Tymgard“ und Modifikation für zyklische Prozesse
- Entwicklung einer auf die Abscheideanlage „Gartek“ zugeschnittene Schaltung zur elektrischen Ansteuerung verschiedener Ventile zur Gasfluss- und Vakuumkontrolle
- Entwicklung einer PC-Schnittstelle zur Benutzerinteraktion
- Herstellen der Kommunikation zwischen PC und Steuerschaltung
- Einbinden des Systems in die Abscheideanlage
- ausführliche Dokumentation der Ansteuerelektronik und des Rechnerprogramms
- Nachweis der Funktionstüchtigkeit des Systems
- Anwendung der Steuerung bei ersten ALD-Prozessen

Kapitel 1

Einleitung

1.1 Allgemeines zur Atomlagenabscheidung

Die Atomlagenabscheidung (Atomic Layer Deposition, ALD) ist ein chemisches Verfahren zur Herstellung dünner Schichten. Sie wurde Ende der 70er Jahre entwickelt zur Fertigung von Dünnfilmelektrolumineszenzdisplays, die großflächige gleichmäßige dielektrische und lumineszente Schichten erfordern. Auch heutzutage wird sie für diesen Zweck noch eingesetzt, allerdings wurde inzwischen ihre Anwendungsmöglichkeit in der Fertigung integrierter Schaltkreise erkannt und die Forschung auf diesem Gebiet stark vorangetrieben [1].

Ein ALD-Prozess besteht im Wesentlichen aus 4 Schritten: 1) Einwirken des ersten Precursors, 2) Spülen der Reaktionskammer, 3) Einwirken des zweiten Precursors, 4) Spülen der Reaktionskammer. Dieser Zyklus wird so oft wiederholt, bis die gewünschte Schichtdicke erreicht wurde. Es lassen sich dadurch einheitliche Schichten mit sehr gut kontrollierbarer Dicke erzeugen, was einen großen Vorteil der ALD gegenüber anderen Verfahren darstellt [1].

Die Anlage „Varian-Gartek“ ist eine CVD-Anlage (CVD: chemical vapour deposition, chemische Gasphasenabscheidung), jetzt soll sie für ALD-Prozesse genutzt werden. An ihr werden die verschiedenen Precursor- und Spülgase über Druckluftventile zu- und abgeschaltet. Deren Ansteuerung erfolgt hier manuell, d. h. die Ventile werden über Schalter am Bedienpult geöffnet bzw. geschlossen. Aufgabe der Prozesssteuereinheit ist nun die automatische Ansteuerung dieser Ventile entsprechend einem vorgegebenen Rezept, welches im Prinzip aus wenigen Schritten besteht, die einen einzelnen Zyklus beschreiben. Das Rezept wird dann je nach gewünschter Schichtdicke mehrere Male wiederholt. Die Anzahl dieser Durchläufe kann bis zu einigen Tausend betragen, da in einem einzelnen Zyklus je nach Schichtmaterial nur 0.1 bis 3 Atomlagen abgeschieden werden.

1.2 Aufgabenbeschreibung

Der erste Teil der Arbeit besteht darin, die vorhandene Prozesssteuereinheit „Tymgard

Process Controller“ wieder in Betrieb zu nehmen [2]. Da diese nicht die nötige Flexibilität hat, ist eine Erweiterung zu entwerfen, die zyklische Prozessrezepte ermöglicht.

Diese Erweiterung ist ein einstellbarer Abwärtszähler, mit dessen Hilfe ein im Tymgard programmiertes Rezept wiederholt nacheinander ausgeführt werden kann. Die Anzahl der Wiederholungen wird dabei durch die Programmierung des Zählers festgelegt. Solche Zählermodule für industrielle Anwendungen existieren zwar schon, sind aber unverhältnismäßig teuer, da die Erweiterung nur als Übergangslösung dienen soll. Aus diesem Grund muss eine eigene einfache Schaltung für diesen Zweck entworfen und aufgebaut werden.

Der zweite Teil, die Hauptaufgabe, ist der Entwurf einer neuen Prozesssteuerung, mit deren Hilfe die Anlage über einen PC gesteuert werden kann. Dazu ist eine Software zur Bearbeitung und Ausführung der Prozessrezepte erforderlich, und eine Steuerschaltung, die entsprechend den Rezepten die Gasfluss- und Vakuumventile schaltet. Der zur Verfügung stehende PC ist eine Windows-2000-Workstation, als Programmiersprache wird C++ gewählt.

Kapitel 2

Das Zählermodul zur Erweiterung des Tymgard-Processcontroller

2.1 Anforderungen und Vorüberlegungen

Der Tymgard ist in der Lage, verschiedene Rezepte zu speichern, mit denen zehn Relaisausgänge und acht analoge Ausgänge angesteuert werden können. Jedes Rezept kann bis zu 99 Schritte enthalten, bei insgesamt 145 Schritten für alle Rezepte zusammen. Das allein ist für zyklische Prozesse allerdings viel zu wenig, da beispielsweise ein Rezept aus zehn Schritten zur Abscheidung von etwa einer halben Atomlage für genügende Schichtdicken mehrere 100 oder gar 1000 mal ausgeführt werden müsste.

Da der Tymgard über externe Eingänge unter anderem zum Neustart des aktuellen Rezeptes verfügt, entstand die Idee, das Rezept nach Ablauf des letzten Schrittes einfach neu zu starten. Ein Zähler überwacht die Anzahl dieser Neustarts und unterbricht bei einer voreingestellten Zahl die Prozessausführung. Der Zähler erhält dabei seinen Takt über eines der Ausgangsrelais des Tymgard. Ein einfacher programmierbarer Abwärts-

zähler, welcher bei einem Zählerstand von Null ein Signal ausgibt, das zur Unterbrechung der Neustarts dient, reicht demnach als Übergangslösung völlig aus.

2.2 Entwurf, Aufbau und Test der Zählerschaltung

Die hier beschriebene Schaltung ist im Bild 5 (Seite 42) im Anhang zu sehen. Grundbaustein des Zählers ist der CMOS-IC 4522 [3]. Das ist ein dezimaler Abwärtszähler, welcher asynchron, also ohne zusätzlichen Taktimpuls vorgeladen und zurückgesetzt werden kann. Die Ausgabe des Zählerstandes erfolgt BCD codiert, ebenso die Eingabe des zu programmierenden Wertes. Der Baustein verfügt außerdem über einen Ausgang zur Nullsignalisierung und über einen Eingang zur Kaskadierung. Vier dieser IC-Bausteine wurden zu einem 4-stelligen Zähler zusammen geschaltet (IC1 bis IC4), mit je einem BCD-zu-7-Segment-Dekoder (IC5 bis IC8, CMOS-IC 4511 [4]) mit nachgeschalteter LED-Anzeige (DIS1 bis DIS4) wird der aktuelle Zählerstand angezeigt. Zur Einstellung des Vorladewertes dienen einfache Dipschalter, welche entsprechend der BCD-Codierung des gewünschten Wertes eingestellt werden müssen. Die Eingänge zum Laden (PRELOAD) und Rücksetzen (MASTERRESET) werden gemeinsam auf je einen Taster geführt.

Beim Zählerstand Null wird zum einen der Zähler deaktiviert, sodass nicht weiter gezählt wird, und zum anderen ein High-Signal ausgegeben, mit dessen Hilfe die Rezept-Neustart-Schaltung deaktiviert wird (links unten im Bild 5 zu sehen). Zur Funktion der Schaltung werden zwei Relaisausgänge des Tymgard benötigt, einer für den Takt des Zählers (Relais 9), der andere zur Erzeugung des Neustartsignals (Relais 10). Die Anschlussbelegungen und die Funktionsweise der Steuersignale des Tymgard sind in dessen Bedienungsanleitung enthalten [2].

Relais 9 schaltet bei Aktivierung den Takteingang gegen 12 V, wodurch der Zähler um Eins erniedrigt wird, es muss also nur im letzten Schritt des Rezeptes angesteuert werden. Relais 10 schaltet bei Aktivierung das RC-Glied aus R18 und C1 gegen 12 V, sodass sich der Kondensator aufladen kann. Fällt es danach wieder ab, so fließt aus dem Kondensator ein Strom über die Emitter-Kollektor-Strecke von Q1 und durch die LED des Optokopplers OK1, wenn der Basisanschluss vom Transistor auf VSS liegt. Das ist der Fall, wenn der Zählerstand nicht Null ist, da die Basis über einen Vorwiderstand direkt vom „0“-Ausgang des niederwertigsten Zählers angesteuert wird. Bei '0' dagegen liegt sie auf 12 V, und der Transistor sperrt. Es kann damit auch kein Strom durch die LED fließen.

Der Kondensator C9 dient hierbei der kurzen Verzögerung der Aktivierung der Optokoppler-LED. Ist diese aktiv, dann ist der ausgangsseitige Phototransistor niederohmig und es werden der Restart- und der AlarmAcknowledge-Eingang des Tymgard über diesen Phototransistor gegen den Masseanschluss (DGND, Digital Ground) geschaltet. Dadurch wird das aktuelle Rezept im Tymgard neu gestartet und der Alarmzustand zurückgesetzt, der bei Rezeptende und nach einem Neustart auftritt. Diese Schaltungsvariante macht es erforderlich, dass das Relais 10 sowohl im ersten wie auch im letzten Schritt des Rezeptes angesteuert wird. Die Dauer dieser beiden obligatorischen Schritte sollte etwa 2 s betragen. Am Anfang des letzten Schrittes wird somit der Zähler dekrementiert und der Kondensator C1 geladen und nach dessen Ende erfolgt der Neustart des Rezeptes und das Zurücksetzen des 'Rezept-Ende-Alarmes'. Im ersten Schritt wird wieder C1 geladen und nach dessen Ende wird der 'Rezept-Neustart-Alarm' zurückgesetzt. Ein Neustart erfolgt hier aber nicht nochmal, da diese vom Tymgard nur akzeptiert werden, wenn gerade kein Rezept ausgeführt wird. Das folgende Taktdiagramm (Bild 1) verdeutlicht den zeitlichen Verlauf der für die Steuerung des Tymgard wichtigen Signale.

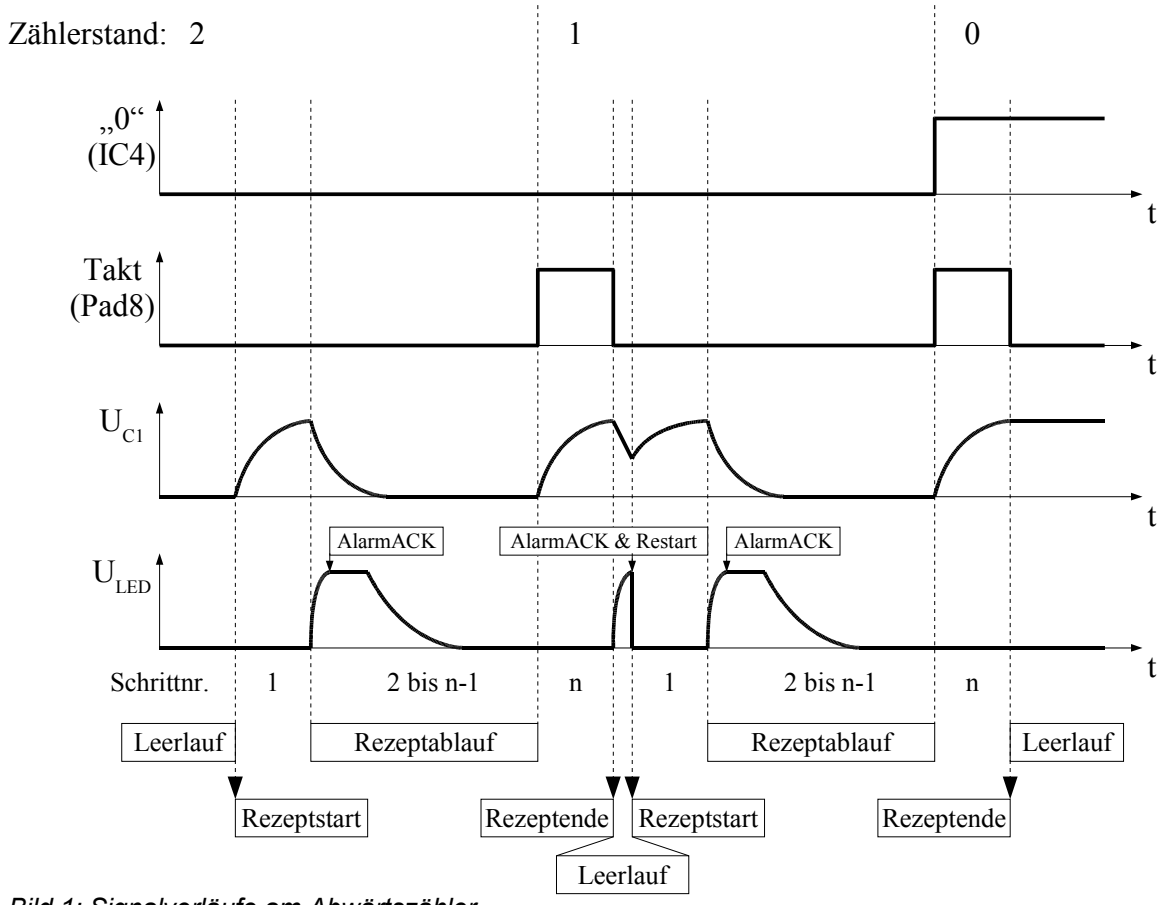

Bild 1: Signalverläufe am Abwärtszähler

Anfangs sollte das Umschalten des Kondensators C1 und die Erzeugung des Taktsignales über ein einziges Relais erfolgen. Allerdings stellte sich heraus, dass die dabei entstehenden relativ flachen Taktflanken (etwa 1 Sekunde für den Low-High-Übergang) von den Zähler-ICs nicht ordentlich erkannt wurden, obwohl diese Schmitttrigger am Takteingang besitzen, die bei eben solchen Flanken eine zuverlässige Triggerung ermöglichen sollten.

Nachdem dieses Problem erkannt wurde, entstand dann die oben erläuterte Schaltungsvariante, welche ordnungsgemäß funktioniert. Alle wichtigen technischen Daten sind auf Seite 53 zu finden.

Ein Rezept hat folgenden prinzipiellen Aufbau:

Tabelle 1: Beispielrezept

Schrittnummer	***Schrittdauer***	***Angesteuerte Ausgänge***
0	1 s (fest)	kein Ausgang (Abbruchschritt)
1	2 s	nur Ausgang 10
2	...	beliebige Ausgänge außer 9 und 10
...	...	beliebige Ausgänge außer 9 und 10
n-1	...	beliebige Ausgänge außer 9 und 10
n	2 s	nur Ausgang 9 und Ausgang 10

Kapitel 3

Die PC-basierte Prozesssteuerung als Ersatz für den Tymgard

3.1 Anforderungen und Vorüberlegungen

Die neue Prozesssteuereinheit soll ein flexibles Arbeiten an der Anlage ermöglichen. Dazu sind mindestens 12 Schaltausgänge für Gasfluss- und Vakuumventile nötig. Damit auch zukünftige Erweiterungen kein Problem darstellen, wird die Ausgangszahl auf 24 erhöht. Als Schnittstelle zwischen Rechner und Schaltmodul wird die serielle gewählt, allerdings werden nicht die seriellen Übertragungskanäle genutzt, sondern nur zwei Statusleitungen der Schnittstelle (DTR, data terminal ready, und RTS, request to send), deren Ausgangswerte direkt von Programmen beeinflussbar sind, d. h. fest auf logisch '1' oder

'0' gesetzt werden können. Auf diesen beiden Leitungen wird das I²C-Bus- Protokoll [5,6] vom Rechner simuliert, um die Relais mit Hilfe der I/O-Expander-Bausteine PCF8574 [7] anzusteuern.

3.2 Entwurf, Aufbau und Test der Interfaceschaltung

Der Schaltplan hierzu ist im Anhang in Bild 8 zu sehen (Seite 44), das daraus entworfene Platinenlayout im Bild 9 (Seite 45).

Zur Wandlung der Signale der Seriellen Schnittstelle dient der IC1 MAX232 [8]. Dieser besitzt zwei Empfänger, die mit den beiden benötigten Signalen für die seriellen Daten (SDA, serial data) und den Takt (SCL, serial clock) vom Rechner beschaltet sind, und zwei Sender, von denen einer zur Rückmeldung des Datensignals an den Rechner genutzt wird (vom Programm im Moment nicht genutzt). SDA wird über DTR übertragen, SCL über RTS. Die Ausgangssignale der Empfänger werden durch die beiden Transistoren Q25 und Q26 invertiert. Der Transistor 26, welcher das Datensignal schaltet, stellt dabei auch den Bus-Charakter der SDA-Leitung wieder her, sodass auch die I/O-Bausteine diese Leitung beeinflussen können.

Die Relais werden über Transistoren geschaltet, deren Basisanschlüsse an die Ausgänge der PCF8574-Bausteine (IC2 bis IC4) angeschlossen sind. Die Ausgänge der IC-Bausteine können nur nach Masse schalten, daher sind sie zusätzlich über Widerstandsnetzwerke mit der Betriebsspannung von 5 V verbunden. Ist ein einem Ausgang zugeordnetes Datenbit logisch '1', ist damit das entsprechende Relais aktiv.

Der Watchdog-IC MAX813 [9] (IC6) dient dazu, bei einem Ausbleiben von Takten auf dem I²C-Bus alle Relais abzuschalten. Dazu ist sein Watchdog-Eingang WDI an die SCL-Leitung angeschlossen. Der Watchdog-Ausgang /WDO führt im Normalfall High-Pegel. Wenn allerdings innerhalb von etwa 1,6 s keine Signaländerung an WDI auftritt, läuft ein interner Zähler über, der /WDO auf Low-Pegel schaltet. Dieser Zähler wird bei jeder Signalflanke an WDI zurückgesetzt. Die restlichen Funktionen dieses Bausteins werden nicht genutzt. Der Ausgang ist dann zu einem Universal-Timer-Baustein, dem NE555 [10] (IC5), weitergeführt, welcher als sogenannter 'missing-pulse-detector' beschaltet ist. Das bedeutet, dass sein Ausgang nach Masse schaltet und damit das Relais 25 aktiviert, wenn über eine mit R5 und C12 einstellbare Zeitspanne (hier etwa 4 Sekunden) sein Triggereingang (Pin 2) dauernd High-Pegel führt. Relais 25 schaltet die Betriebsspannung für die restlichen Ausgangsrelais. Tritt ein Low-Impuls an diesem Pin 2

auf, so schaltet der Ausgang nach 5 V und das Relais 25 wird inaktiv. Im Zusammenspiel mit dem MAX813 bedeutet das Folgendes: Wenn Daten auf dem Bus übertragen werden, ist der Watchdog-Ausgang dauernd auf High-Pegel und das Relais 25 aktiv. Sobald aber für etwa 1,6 Sekunden die Taktsignale ausbleiben, schaltet der NE555 das Relais ab. Erst nachdem für 4 s der Triggereingang High-Pegel führt, wird es wieder aktiviert.

Der Einschaltvorgang der Schaltung ist damit auch kontrolliert, da der NE555 erst nach 4 s die Betriebsspannung für die restlichen Relais aktiviert, aber vor Ablauf dieser Zeit der Watchdog den Triggereingang auf Low zieht und damit das Einschalten des Relais 25 verhindert. Die genauen Signalverläufe zeigt das folgende Diagramm (Bild 2).

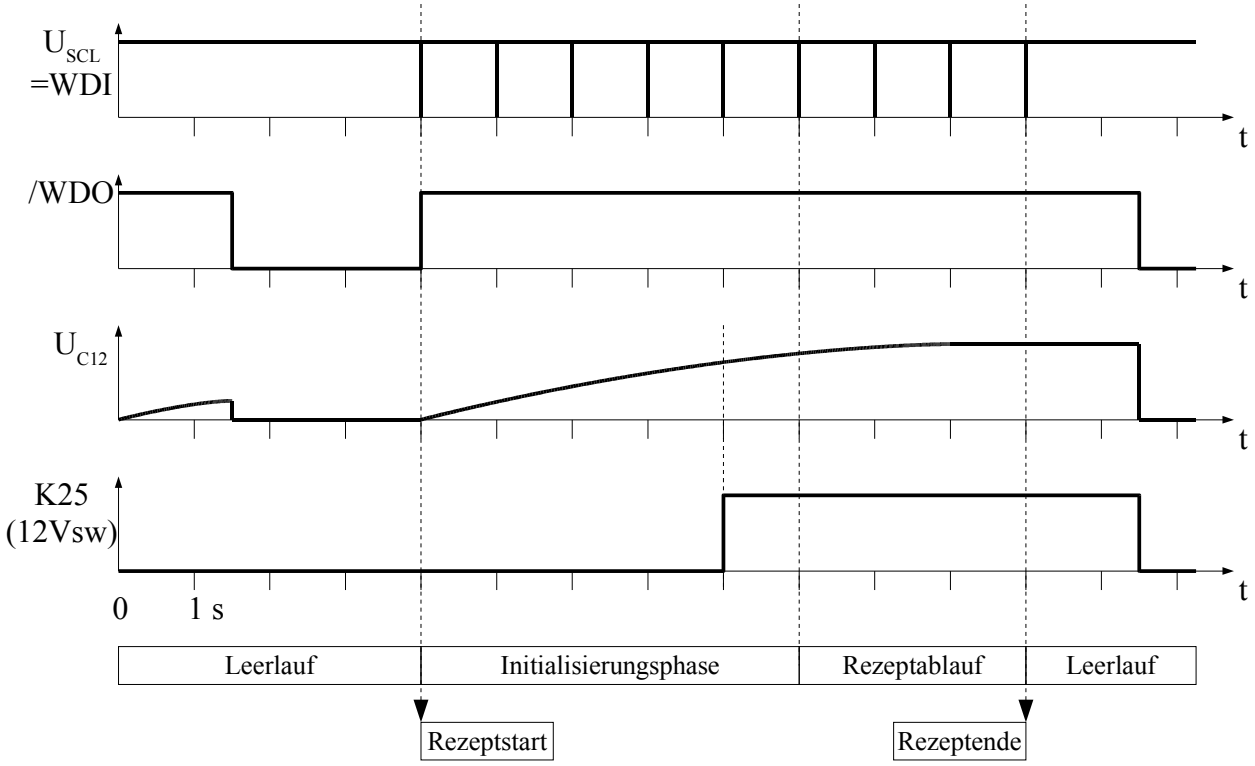

Bild 2: Signalverläufe am Watchdog und am Timer-IC

Die Betriebsspannung wird zum Zeitpunkt 0 eingeschaltet. /WDO ist nur für 1,6 s High, da an WDI keine Signalwechsel stattfinden. Da 1,6 s Ladezeit für C12 nicht ausreichen, um ihn auf 66% (etwa 3,3 V) der Betriebsspannung aufzuladen, schaltet K25 keine 12V für die Ausgangsrelais. Nach dem Starten eines Rezeptes werden zuerst fünf mal drei '0'-Bytes ausgegeben. Dadurch werden alle Ausgänge der IC-Bausteine 2 bis 4 auf '0' gesetzt. Während dieser Initialisierungsphase kann sich C12 aufladen, und bei etwa 3,3 V aktiviert IC5 K25 und schaltet damit die 12V Versorgungsspannung für die Ausgangsrelais zu. Mit der sechsten Übertragung wird dann der erste programmierte Schaltzustand

eingestellt. Bei Rezeptende werden zuletzt wieder drei '0'-Bytes ausgegeben, danach erfolgt keine Übertragung und somit auch keine Änderung an WDI mehr. Der Watchdog schaltet daher nach 1,6 s /WDO nach Masse, C12 wird entladen und K25 wieder abgeschaltet. Beim Start schaltet Windows zwar die benutzten Steuerleitungen der seriellen Schnittstelle zwischen '0' und '1' hin und her, allerdings nicht über einen Zeitraum von 4 s, sodass auch dabei die Ausgangsrelais keine Versorgungsspannung erhalten.

Über die Anschlüsse X27 und X28 werden die Betriebsspannung und die Bus-Signale herausgeführt, sodass die Schaltung um weitere Ausgänge erweitert werden kann. Damit auch F1 automatisch gesteuert werden kann, ist ein Umbau des Schalters nötig. Den Zustand danach zeigt Bild 14 (Seite 50).

Alle wichtigen technischen Daten sind auf Seite 53 zu finden.

3.3 Die Software

Die Softwarekomponente unterteilt sich in 2 eigenständige Programme: Einen Editor zum Bearbeiten und ein Programm zur Ausführung von Rezepten.

Ein Rezept besteht aus einzelnen Schritten, deren Dauer in Sekunden festgelegt wird und denen aktive Ausgänge zugeordnet werden, analog der Funktionalität des Tymgard. Allerdings gibt es hier keinen separaten Abbruchschritt mehr, sondern dieser ist grundsätzlich so festgelegt, dass alle Ausgänge inaktiv sind. Die Schritte werden nacheinander abgearbeitet, und das gesamte Rezept für eine einstellbare Anzahl wiederholt ausgeführt. Es existiert ein spezieller Wiederholungsschritt, welcher an beliebiger Stelle im Rezept auftreten kann. Er bewirkt, dass alle in der Reihenfolge vor ihm liegenden Schritte bis zum vorhergehenden Wiederholungsschritt bzw. Rezeptanfang für eine festlegbare Anzahl wiederholt werden. Ein Wiederholungsschritt als erster Schritt wird bei der Ausführung des Rezeptes ignoriert.

Das Editor- und das Ausführungsprogramm müssen sich zusammen mit der Konfigurationsdatei Config.ini in einem Verzeichnis befinden, das Verzeichnis selbst an beliebiger Stelle. Die Rezeptdateien können an einem anderen Ort gespeichert werden.

Der Inhalt einer beispielhaften Konfigurationsdatei ist im Folgenden zu sehen, alle Einstellmöglichkeiten sind darin bereits erläutert.

```
################################################
# Konfigurationsdatei zur Prozessablaufsteuerung #
################################################
# Kommentare müssen mit '#' beginnen !
```

```
# Bei den Schlüsselwörtern auf Groß-/Kleinschreibung achten !

# Serielle Schnittstelle, an dem die Steuerplatine angeschlossen ist:
ComPort=COM2

# Befehl, mit dem das Prozessausführungsprogramm gestartet wird:
PX_Command=C:\Wichtiges\Studienarbeit\Release\PX.exe

# Folgende Zeilen ordnen den Spalten (Kanälen) im Programm die Ausgangsrelais
# und die symbolischen Namen zu.
# Syntax: Kanal<wert1>=Relais<wert2>=[symbolischer Name]
# wert1 und wert2 sind Ganzzahlen zwischen 1 und 24
# der symbolische Name sollte nicht länger als 10 Zeichen sein
Kanal1=Relais1=NH3
Kanal2=Relais2=NH3Vent
Kanal3=Relais3=Ar
Kanal4=Relais4=ArVent
Kanal5=Relais5=H2
Kanal6=Relais6=O2
Kanal7=Relais7=H2O
Kanal8=Relais8=Prec_LDS1
Kanal9=Relais9=Prec_LDS2
Kanal10=Relais10=D2D3_Close
Kanal11=Relais11=D1
Kanal12=Relais12=F1
Kanal13=Relais13=K13
Kanal14=Relais14=K14
Kanal15=Relais15=K15
Kanal16=Relais16=K16
Kanal17=Relais17=K17
Kanal18=Relais18=K18
Kanal19=Relais19=K19
Kanal20=Relais20=K20
Kanal21=Relais21=K21
Kanal22=Relais22=K22
Kanal23=Relais23=K23
Kanal24=Relais24=K24
```

In den Rezeptdateien werden für jeden Schritt die symbolischen Namen gespeichert. Werden diese in der Config.ini geändert, so treten mit älteren Rezepten Fehler auf, da die alten Namen dann nicht mehr zugeordnet werden können. Wird aber nur die Zuordnung der Namen zu Kanälen oder Relais geändert, die Namen selbst aber alle belassen, treten keine Probleme auf.

3.3.1 Der Editor

Der Rezepteditor, wie in Bild 3 gezeigt, besteht im Wesentlichen aus einer Tabelle, welche die Rezeptdaten darstellt und einer Anzahl von Buttons zur Aktionsauswahl.

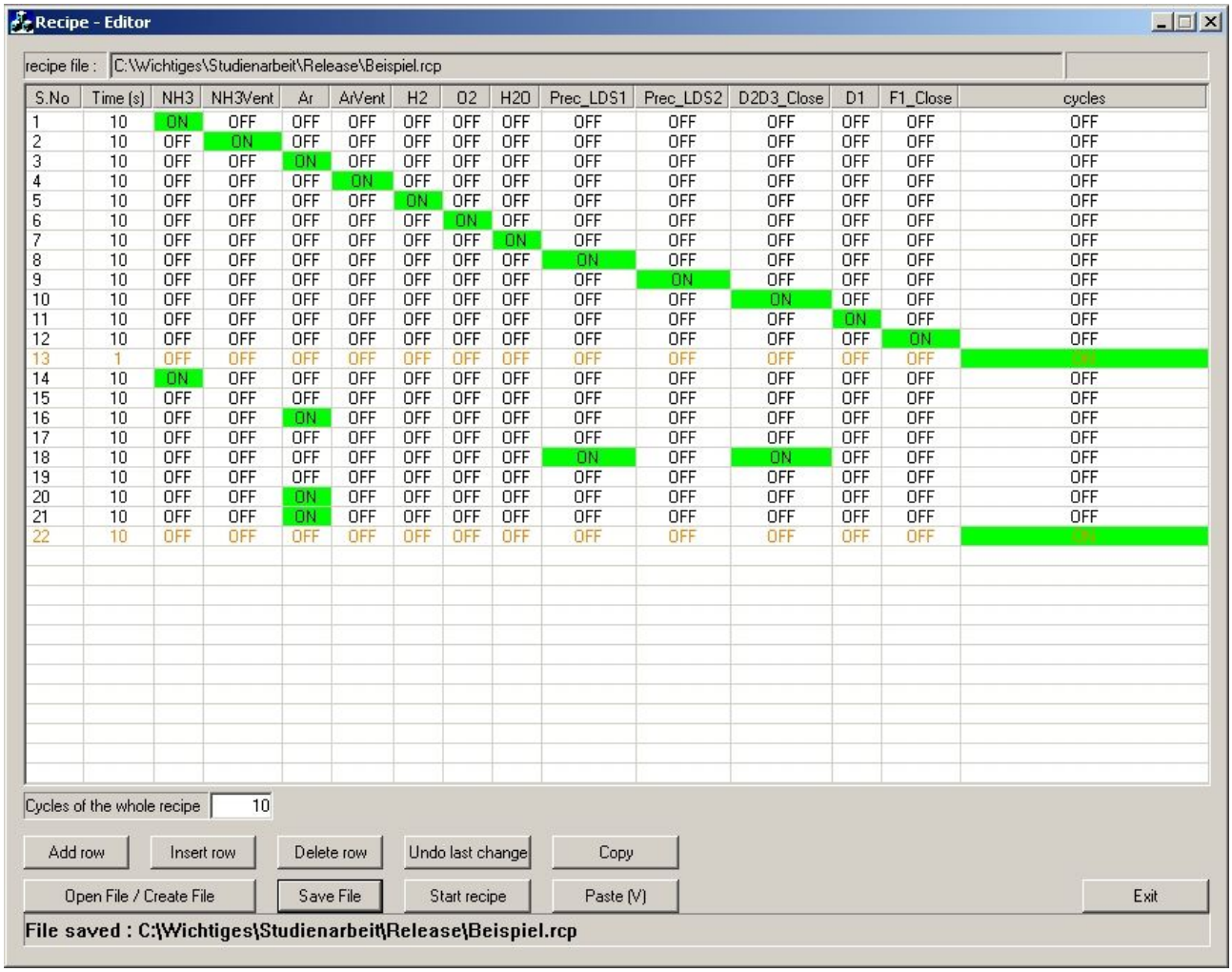

Bild 3: Das Editorfenster

Der Tabellenkopf beinhaltet (von links nach rechts) die Schrittnummer, die Schrittdauer, die Namen der angesteuerten Ventile und ein Spalte mit dem Titel „cycles“. Die Schrittnummer kann nicht verändert werden. Die Werte in den einzelnen Zellen in der Spalte „Schrittdauer“ der Tabelle können nach einem Doppelklick auf die gewünschte Zelle bearbeitet werden. Sie stellen die Dauer des Schrittes in Sekunden dar. Die kleinste Zeitspanne beträgt 1 s, daher sind hier nur ganze Zahlen erlaubt. Die angesteuerten Ventile, also die Ausgänge der Schaltung, können ebenfalls mit einem Doppelklick aktiviert bzw. deaktiviert werden. Ein grüner Zellhintergrund bedeutet dabei, dass der entsprechende Ausgang aktiv ist. Wird die Zelle „cycles“ in einer Zeile aktiviert, so werden

alle anderen Ausgänge der Zeile deaktiviert und dieser Schritt zum Wiederholungsschritt, dessen Anzahl an Wiederholungen in der Zelle „Schrittdauer“ festgelegt wird.
Ganz oben im Fenster befindet sich eine Zeile, die den Namen und Pfad der aktuellen Rezeptdatei anzeigt. Rechts daneben erscheint ein „modified“, wenn an der Datei seit dem letzten Speichern Änderungen vorgenommen wurden. Unterhalb der Tabelle werden bei „cycles of the whole recipe“ die Gesamtwiederholungen des Rezeptes festgelegt. Die Statusleiste ganz unten gibt Auskunft über das Öffnen oder Schließen von Rezeptdateien. Die Buttons haben folgende Funktionen :

Tabelle 2: Buttons des Editorfensters

Button	***Funktion***
Add row	Hängt an das Ende des Rezeptes eine einzelne Zeile an
Insert row	Fügt vor der aktuell markierten eine neue Zeile ein
Delete row	Löscht die aktuell markierte Zeile(n)
Undo last change	Macht die zuletzt vorgenommene Änderung rückgängig; es kann immer nur eine einzige Änderung rückgängig gemacht werden
Copy	Kopiert die ausgewählten Schritte in eine Zwischenablage
Paste (V)	Fügt die Schritte in der Zwischenablage vor dem aktuell markierten Schritt ein; der Inhalt der Ablage geht dabei nicht verloren
Open File/Create file	Öffnet ein Dialogfenster zum Öffnen einer Rezeptdatei; es kann dort entweder eine vorhandene Datei ausgewählt werden oder eine neue angelegt werden, indem ein nicht vorhandener Dateiname eingegeben wird
Save file	Öffnet ein Dialogfenster zum Speichern der Rezeptdatei; es kann ebenfalls entweder eine vorhandene Datei ausgewählt werden oder unter einem neuen Namen gespeichert werden, indem ein nicht vorhandener Dateiname eingegeben wird
Start recipe	Speichert das Rezept, startet das Ausführungsprogramm mit dem Rezept und führt es auch sofort aus
Exit	Beendet das Programm; war das Rezept nicht gespeichert, erfolgt noch eine Abfrage

Ein neues Rezept würde beispielsweise folgendermaßen erstellt: Zuerst wird eine neue Datei angelegt, indem im Öffnen-Dialogfenster ein nicht vorhandener Dateiname eingeben wird. Danach wird eine gewünschte Anzahl an Schritten mit „Add row“ hinzugefügt. Die Zeiten der Schritte können nun nach einem Doppelklick auf die Werte angepasst werden,

ebenso die aktiven Ausgänge. Beim Speichern wird automatisch der zuvor gewählte Dateiname eingetragen, es kann aber auch ein neuer vergeben werden. Dabei bleibt allerdings die zu Beginn angelegte Datei erhalten, natürlich ohne Inhalt.

3.3.2 Das Ausführungsprogramm

Die Oberfläche des Ausführungsprogrammes ist in Bild 4 gezeigt. Die erste Zeile zeigt den Namen und den Pfad der aktuell geöffneten Rezeptdatei an.

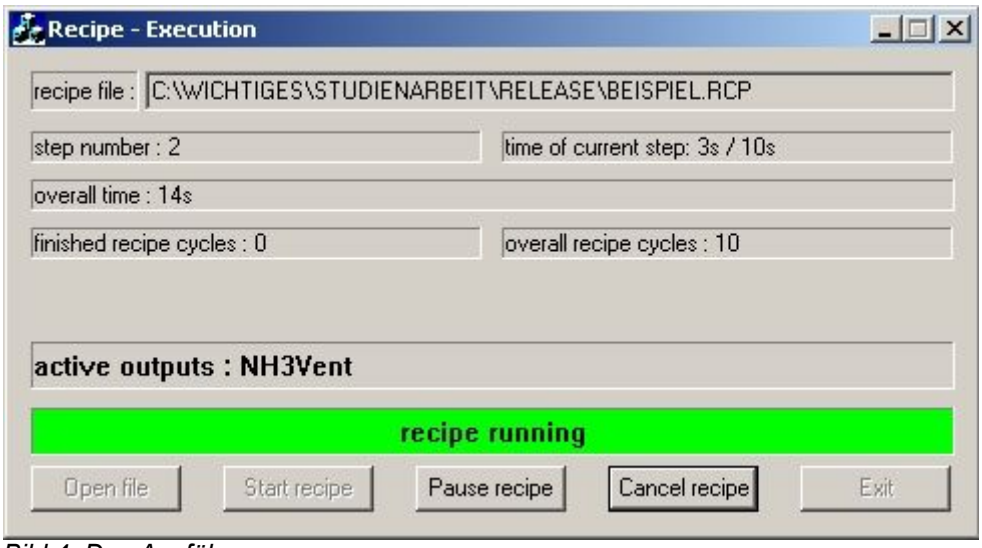

Bild 4: Das Ausführungsprogramm

Die acht darunterliegenden Informationsfelder haben folgende Bedeutungen:

Tabelle 3: Informationsfelder der Programmoberfläche

Feldname	***Beschreibung***
step number	Nummer des aktuellen Schrittes im Rezept
time of current step	bereits vergangene Zeit / Gesamtzeit des aktuellen Schrittes
overall time	momentane Laufzeit des Rezeptes
finished recipe cycles	abgeschlossene Durchläufe des Rezeptes
overall recipe cycles	Gesamtzahl der Rezeptdurchläufe
active outputs	Namen der momentan aktiven Ausgänge

Darunter folgt eine Statuszeile, die über die Aktivitäten des Rezeptes informiert (laufend, pausiert, beendet oder abgebrochen).

Die Buttons haben folgende Funktionen:

Tabelle 4: Buttons des Ausführungsprogrammes

Button	***Funktion***
Open file	Öffnet ein Dialogfenster zum Öffnen einer vorhandenen Rezeptdatei
Start recipe	Startet das geöffnete Rezept
Pause recipe	Pausiert das laufende Rezept; während dieser Zeit sind alle Ausgänge deaktiviert
Cancel recipe	Bricht die Ausführung des laufenden Rezeptes ab; alle Ausgänge werden deaktiviert
Exit	Beendet das Programm

Je nach aktuellem Status sind einige der Buttons deaktiviert, damit ein versehentliches Klicken ohne Folgen bleibt. Bei laufendem oder pausiertem Rezept kann daher kein neues geöffnet werden oder das Programm beendet werden, sondern das aktuelle muss erst abgebrochen werden.

4 Auswertung

Der Tymgard kann mit seiner Erweiterung zur Ausführung von Rezepten mit 97 Schritten, welche bis zu 9999 mal wiederholt werden können, benutzt werden. Der erste und der letzte Schritt dienen immer der Steuerung der Zählerschaltung und stehen für den Prozess selbst nicht zur Verfügung. Es muss außerdem beachtet werden, dass nach Prozessende der Kondensator C1 geladen bleibt. Das hat zur Folge, dass das aktuelle Rezept sofort wieder neu gestartet wird, sobald man den Zähler mit einem neuen Wert lädt. Auch die Programmierung der Anzahl der Durchläufe ist aufgrund der nötigen Umrechnung in einen Binärwert nicht unbedingt bedienerfreundlich. Aber die Schaltung genügt den Anforderungen von zyklischen Prozessen, für welche die Anlage vorgesehen ist. Allerdings kann der Tymgard damit nur noch acht Ventile steuern, für eine ausreichend flexible Steuerung jedoch sind mindestens 14 Ausgänge nötig. Diese Version kann daher nur als Behelfslösung angesehen werden.

Die im Kapitel 3 beschriebene Steuerung stellt 24 programmierbare Ausgänge zur Verfügung. Über die dazugehörige Software können diese mit Hilfe von nahezu beliebigen Rezepten angesteuert werden, die Anlage „Varian Gartek“ kann damit im gewünschten

Umfang automatisch arbeiten. Die Erweiterung der Ausgangszahl auf bis zu 64 ist ebenfalls möglich, dazu ist allerdings auch eine Änderung der Software nötig. Das kleinste einstellbare Zeitintervall ist 1 Sekunde, was für die geplanten Prozesse ausreicht. Bei Computerabsturz oder wenn durch andere Fehler keine Signale mehr an der Interfaceplatine ankommen, werden nach einer Wartezeit von etwa 1,6 s alle Ausgänge deaktiviert, um die Anlage in einen sicheren Zustand zu versetzen. Der Rezeptaufbau ist ähnlich dem vom Tymgard gestaltet, d. h. ein Einzelschritt ist gekennzeichnet durch seine Nummer, seine Dauer und die aktiven Ausgänge. Es existiert zusätzlich ein spezieller Wiederholungsschritt, der es erlaubt, einzelne Sequenzen innerhalb eines Rezeptes zu wiederholen. Der Umfang der Rezepte ist nur vom Speicher des Rechners, auf dem die Software läuft, abhängig. Dieser ist auch für komplexe Rezepte in der Regel völlig ausreichend.

5 Zusammenfassung

Der Tymgard Process Controller wurde durch einen programmierbaren Abwärtszähler erweitert und kann damit für zyklische Rezepte mit bis zu 97 Einzelschritten und 9999 Wiederholungen eingesetzt werden. Ein Schritt ist dabei gekennzeichnet durch seine Dauer und die aktiven Ausgänge. Mit dem Tymgard lassen sich 8 Ventile der Anlage steuern.

Das neu entworfene Steuerungsmodul besitzt 24 Ausgangsrelais zum Schalten von Gasfluss-, Vakuumventilen oder anderen Einrichtungen, welche über eine eigens dafür geschriebene Software kontrolliert werden. Die Rezepte sind ähnlich denen im Tymgard aufgebaut und werden in Dateien gespeichert. Ihr Umfang ist nur vom Speicher des Rechners, auf dem die Software läuft, abhängig, wobei auch sehr umfangreiche Rezepte kein Problem sind.

6 Anhang

Hier sind die Quelltexte des Rezepteditors und des Ausführungsprogrammes abgedruckt zusammen mit Erläuterungen zur Implementierung. Es folgen danach die Schaltpläne des Abwärtszählers, der Prozesssteuereinheit und der Verdrahtungsplan des umgebauten Steuerteiles des Schaltschrankes der Anlage „Varian Gartek“ und Fotografien der genannten Geräte.

6.1 Quelltext des Rezepteditors

Das Hauptelement der Oberfläche des Programms ist ein ListControl-Steuerelement, welches hier als Tabelle zur Anzeige des Rezeptes genutzt wird. Auf sie wird über die Variable *m_Schrittliste* zugegriffen. Das Rezept wird in zwei Feldern gespeichert, in *Schrittdauer* die Zeiten der einzelnen Schritte und in *Kanalwert* die angesteuerten Ausgänge, wobei jedes Bit von *Kanalwert* einem einzelnen Ausgang entspricht. Diese werden automatisch mit Hilfe der Funktion *Felder_erweitern* den Anforderungen des Rezeptes angepasst und in jedem Fall größer als das Rezept gehalten.

Nach dem Starten des Programms liest dieses zuerst die Konfigurationsdatei *Config.ini* ein, in welcher die Bezeichner der einzelnen Kanäle und deren Zuordnung zu den Relaisausgängen stehen. Die Bezeichner werden in die Tabelle als Spaltenköpfe eingefügt und die Zuordnung im Feld *Bezeichner* gespeichert.

Nach dem Öffnen einer Rezeptdatei wird diese eingelesen und der Inhalt sowohl in die Felder als auch in die Tabelle eingetragen. Bei Änderungen müssen stehts Tabelle und Felder aktualisiert werden. Beim Speichern werden nur die Felder als Datenquelle benutzt.

Für die Undo-Funktion werden zwei zusätzliche Felder angelegt, *Schrittdauer_bak* und *Kanalwert_bak*. Vor jeder Änderung werden die Inhalte der Datenfelder dorthin kopiert, und beim Betätigen des Undo-Buttons aus den *bak*-Feldern wiederhergestellt. Das wird von der Funktion *backup* erledigt, welche in Abhängigkeit vom Parameter entweder sichert oder wiederherstellt.

Für das Kopieren und Einfügen werden ebenfalls zwei Felder angelegt, *Schrittdauer_buffer* und *Kanalwert_buffer*. Diese sind jeweils nur so groß, dass sie alle markierten Elemente fassen.

Es folgt jetzt eine Übersicht über die deklarierten Variablen und Methoden, entnommen aus der Headerdatei PSTGDlg.h:

```
HICON m_hIcon;

ULONG Zeilenwert;
CString vollstDateiname, Dateiname, Pfad;
CStdioFile Konfdatei, PAPdatei;
CFileException error;
CString Bezeichner[25];
int *Schrittdauer;
```

```
ULONG *Kanalwert;
int maxSchritte, aktSchrittanzahl;
int i;
int Gesamtdurchlauf;
int nItem, nSubItem;
CString Fehler, FehlerTitel;
CString pxcmd;
CString Konfpath;
int *Schrittdauer_bak;
ULONG *Kanalwert_bak;
int *Schrittdauer_buffer;
ULONG *Kanalwert_buffer;
int bufferlaenge;
int Schrittanzahl_bak;

bool gespeichert;

bool CPSTGDlg::KonfigurationLesen();
bool CPSTGDlg::DateiEinlesen();
bool CPSTGDlg::DateiSchreiben();
bool CPSTGDlg::backup(bool save);
afx_msg void OnOK();
afx_msg void OnCancel();
int Felder_erweitern(int elemente);
```

Der folgende Code stammt aus der Implementierungsdatei PSTGDlg.cpp. In der Methode CPSTGDlg::INITDIALOG werden einige der Variablen mit Werten initialisiert:

```
Gesamtdurchlauf = 1;
Zeilenwert = 0;
Schrittdauer = NULL;
Kanalwert = NULL;
maxSchritte = 0;
Schrittdauer_bak = NULL;
Kanalwert_bak = NULL;
Schrittdauer_buffer = NULL;
Kanalwert_buffer = NULL;
Schrittanzahl_bak = 0;
aktSchrittanzahl = 0;
gespeichert = TRUE;

m_Undo.EnableWindow(FALSE);

CFont font;
VERIFY(font.CreatePointFont(140, "Arial", NULL));
m_CFileinfo.SetFont(&font, TRUE);

//Stil der ListControl festlegen
m_Schrittliste.SetExtendedStyle(LVS_EX_FULLROWSELECT | LVS_EX_GRIDLINES);

//EditBox verstecken
m_EditEntry.ShowWindow(SW_HIDE);
```

Die Behandlungsroutinen für die Buttons zum Öffnen, Speichern und Starten des Rezeptes und dem Beenden des Programms sind im Folgenden abgedruckt. Beim Öffnen und Speichern wird jeweils ein vorgefertigtes Dialogfenster zur Auswahl eine Datei benutzt (CFileDialog). Zum Starten des Ausführungsprogrammes wird die zugehörige Befehlszeile

aus der *Config.ini* verwendet.

```
//Klick auf den Open-Button
void CPSTGDlg::OnOeffnen()
{
int antwort = 7;
if (!gespeichert) antwort = MessageBox("Recipe not saved !\n Do you want
                  to save it ?","Warning",MB_ICONWARNING|MB_YESNOCANCEL);
//6 = Yes, 7 = No, 2 = Cancel

      CFileDialog DateiDialog(
            TRUE,  // TRUE = Datei oeffnen
            "rcp",  // Standardwert für anzuhängende Dateierweiterung
            NULL,  // Standardwert für das Feld des Dateinamen
            NULL,  // OPENFILENAME-Flags
            "Recipe file (*.rcp)|*.rcp|all files|*.*|", // Filterregeln
            NULL); // Verweis auf Elterndialog

      switch (antwort)
      {
      case 6 :
            // Yes -> alte Datei speichern, danach Fenster zum Öffnen der
            // neuen anzeigen
            OnSpeichern();
      case 7:
            //No -> neue Datei öffnen
            DateiDialog.DoModal(); //Dialog öffnen
            vollstDateiname = DateiDialog.GetPathName();
            Dateiname = DateiDialog.GetFileName();
            Pfad = DateiDialog.GetPathName();

            if (!vollstDateiname.IsEmpty())
            {
                  //Felder löschen sofern existent
                  free(Schrittdauer);
                  Schrittdauer = NULL;
                  free(Kanalwert);
                  Kanalwert = NULL;

                  //Felder allokieren
                  maxSchritte = 0;
                  if (!Felder_erweitern(50))
                        OnBeenden();
                  else
                  {

                        //Liste löschen
                        m_Schrittliste.DeleteAllItems();

                        if (!DateiEinlesen())
                        {
                              //Einlesen fehlgeschlagen -> Felder löschen
                              free(Schrittdauer);
                              Schrittdauer = NULL;
                              free(Kanalwert);
                              Kanalwert = NULL;

                              m_Fileinfo.LoadString(IDS_DateifehlerPAP);
                        }
                        else
                        {
                              m_Fileinfo = "File opened : " + vollstDateiname;
                              gespeichert = TRUE;
                              m_EditDateiname = vollstDateiname;
                              m_GDL = Gesamtdurchlauf;
```

```
                              char buffer[20] = "";
                              int i, j;

                              //Feldinhalt in die Schrittliste eintragen
                              for (i = 0; i<aktSchrittanzahl; i++)
                              {
                                    _itoa(i+1, buffer, 10);
                                     //Schrittnummer eintragen
                                    m_Schrittliste.InsertItem(i,buffer);

                                    _itoa(Schrittdauer[i], buffer, 10);
                                    //Schrittdauer eintragen
                                    m_Schrittliste.SetItemText(i,1, buffer);
                                    //Kanalwerte eintragen
                                    for (j = 0; j<25; j++)
                                    {
                                    if (Kanalwert[i] & (ULONG)pow(2, j))
                                     m_Schrittliste.SetItemText(i,j+2, "ON");
                                    else
                                     m_Schrittliste.SetItemText(i,j+2, "OFF");
                                    }

                              }
                        }
                  }
            }

            if (gespeichert) m_modified = ""; else m_modified = "modified";
            UpdateData(FALSE);  //Daten im Dialog aktualisieren
            break;
      case 2:
            //Cancel -> Nix machen
            break;
      default:
            break;
      }
}

//Klick auf den Save-Button
void CPSTGDlg::OnSpeichern()
{
      CFileDialog DateiDialog(
            FALSE,  // FALSE = Datei speichern
            "rcp",  // Standardwert für anzuhängende Dateierweiterung
            Dateiname,  // Standardwert für das Feld des Dateinamen
            NULL,  // OPENFILENAME-Flags
            "Recipe file (*.rcp)|*.rcp|all files|*.*|", // Filterregeln
            NULL); // Verweis auf Elterndialog

      DateiDialog.DoModal(); //Dialog öffnen

      vollstDateiname = DateiDialog.GetPathName();
      Dateiname = DateiDialog.GetFileName();
      Pfad = DateiDialog.GetPathName();


      if (!vollstDateiname.IsEmpty())
      {

            if(!DateiSchreiben())
            {
                  m_Fileinfo = "Error writing file !";
                  gespeichert = FALSE;
            }
            else
            {
```

```
                  m_Fileinfo = "File saved : " + vollstDateiname;
                  gespeichert = TRUE;
            }
            m_EditDateiname = vollstDateiname;
            UpdateData(FALSE);  //Daten im Dialog aktualisieren
      }
      if (gespeichert) m_modified = ""; else m_modified = "modified";
      UpdateData(FALSE);
}

//Rezept ausführen
void CPSTGDlg::OnStarten()
{
      CString argument = "";
      argument = "-file:" + vollstDateiname + " -exec";
      if (!vollstDateiname.IsEmpty())
            if (DateiSchreiben())
            {
                  m_EditDateiname = vollstDateiname;
                  m_Fileinfo = "File saved : " + vollstDateiname;
                  UpdateData(FALSE);  //Daten im Dialog aktualisieren

                  gespeichert = TRUE;

                  //Starten des Ausführungsprogramms
                  int error =(int)ShellExecute( this->m_hWnd,"open",pxcmd,
                                          argument,Konfpath,SW_SHOW);
                  if(error < 33)
                  {
                        Fehler.FormatMessage("Error %1!i! starting
                                          PX.exe",error);
                        MessageBox(Fehler,"Error",MB_OK);
                  }
            }
            else
            {
                  m_Fileinfo = "Error writing file !";
                  UpdateData(FALSE);  //Daten im Dialog aktualisieren
                  gespeichert = FALSE;
            }
      if (gespeichert) m_modified = ""; else m_modified = "modified";
      UpdateData(FALSE);

}

//Programm beenden
void CPSTGDlg::OnBeenden()
{
      int antwort = 7;
      if (!gespeichert) antwort = MessageBox(   "Recipe not saved !\n Do you
            want to save it ?","Warning",MB_ICONWARNING|MB_YESNOCANCEL);
      //6 = Yes, 7 = No, 2 = Cancel

      switch (antwort)
      {
      case 6:
            OnSpeichern();
      case 7:
            free(Schrittdauer);
            free(Kanalwert);
            free(Schrittdauer_bak);
            free(Kanalwert_bak);
            free(Schrittdauer_buffer);
            free(Kanalwert_buffer);
            CDialog::OnOK();
            break;
```

```
	case 2:
		break;
	default:
		break;
	}
}
```

Die folgende Methode behandelt den Doppelklick auf ein Element der Tabelle. Erfolgt dieser in der Spalte „Time“, so wird eine EditBox an diese Stelle platziert, die das Verändern der Zeit erlaubt. Wird auf einen Kanalwert doppelt geklickt, wird dieser umgeschaltet (von „OFF“ auf „ON“ und umgekehrt). Ein Doppelklick in der Spalte „cycle“ bewirkt außer der Umschaltung zusätzlich eine Deaktivierung aller anderen Kanäle.

```
//wird bei Doppelklick auf ein Element der Tabelle aufgerufen
//positioniert die EditBox über dem Element, auf das doppelt geklickt wurde
//die Größe der EditBox wird an das ListControl-Element angepasst
void CPSTGDlg::OnDblclkSchrittliste(NMHDR* pNMHDR, LRESULT* pResult)
{
	LPNMITEMACTIVATE temp = (LPNMITEMACTIVATE) pNMHDR;

	nItem = temp->iItem; //Zeilennummer
	nSubItem = temp->iSubItem;//Spaltennummer

	backup(TRUE);

	gespeichert = FALSE;
	if (gespeichert) m_modified = ""; else m_modified = "modified";
	UpdateData(FALSE);

	if(!(nSubItem == 1) || nItem == -1)
	//welche Elemente können so bearbeitet werden
	{
		if(nSubItem < 2) //Doppelklick auf Kanalwerte ?
		return ;
		CString str;
		if (nSubItem == 26) //Doppelklick auf den "Wiederholungskanal" ?
		{
			str = m_Schrittliste.GetItemText(nItem,26);
			if (str == "ON")
			{
				m_Schrittliste.SetItemText(nItem, 26, "OFF");
				Kanalwert[nItem] = 0;
			}
			if (str == "OFF")
			{
				m_Schrittliste.SetItemText(nItem, 26, "ON");
				Kanalwert[nItem] = 16777216;
				for (i=2; i<26; i++)
					m_Schrittliste.SetItemText(nItem, i, "OFF");
			}


		}
		else
		{
			str = m_Schrittliste.GetItemText(nItem ,26);
			//Wiederholungsschritt ?
			if (str == "OFF") //wenn nein, dann Änderungen zulassen
			{
				str = m_Schrittliste.GetItemText(nItem,nSubItem);
				if (str == "ON")
```

```
                        {
                              m_Schrittliste.SetItemText(   nItem, nSubItem,
                                                            "OFF");
                              Kanalwert[nItem] -= (ULONG)pow(2, nSubItem - 2);
                        }
                        if (str == "OFF")
                        {
                              m_Schrittliste.SetItemText(   nItem, nSubItem,
                                                            "ON");
                              Kanalwert[nItem] += (ULONG)pow(2, nSubItem - 2);
                        }
                  }
            }
            return ;
      }
      Invalidate();
      //Text des ausgewählten Elements holen
      CString str = m_Schrittliste.GetItemText(nItem,nSubItem);

      CRect rect_item, rect_listctrl, rect_dialog;
      //Rechteck des ausgewählten Elements holen
      m_Schrittliste.GetSubItemRect (temp->iItem,temp->iSubItem,LVIR_BOUNDS,
                                    rect_item);
      //Rechteck der ListControl holen
      ::GetWindowRect(temp->hdr.hwndFrom,rect_listctrl);
      //Rechteck des ganzen Dialogfensters holen
      GetClientRect(&rect_dialog);
      ClientToScreen(&rect_dialog);

      int x=rect_listctrl.left-rect_dialog.left;
      int y=rect_listctrl.top-rect_dialog.top;

      if(nItem != -1)
      m_EditEntry.SetWindowPos (    &wndTop, rect_item.left + x + 6,
                                    rect_item.top + y + 2,
                                    rect_item.right-rect_item.left - 4,
                                    rect_item.bottom-rect_item.top - 1,NULL);
      //EditBox anzeigen
      m_EditEntry.ShowWindow(SW_SHOW);
      //und ihr den Fokus geben
      m_EditEntry.SetFocus();
      //Rechteck um die EditBox zeichnen
      ::Rectangle(::GetDC(    temp->hdr.hwndFrom), rect_item.left,
                              rect_item.top-1, rect_item.right+1,
                              rect_item.bottom);
      //Text der EditBox auf den Wert setzen, der im ListControl-Element steht
      m_EditEntry.SetWindowText(str);

      *pResult = 0;
}
```

Die Enter-Taste beendet standardmäßig das Programm. Das wird durch folgende Methode verhindert. Außerdem wird beim Drücken auf Enter innerhalb der EditBox der Wert in die Liste übernommen.

```
//Behandlung der Enter-Taste
//hier nur, wenn sie in der EditBox gedrückt wird
//standardmäßig wird damit das Dialogfenster geschlossen, das wird so
//verhindert
void CPSTGDlg::OnOK()
{

      CWnd* pwndCtrl = GetFocus();
```

```
        // control ID der Control holen, welche den Fokus hat
        int ctrl_ID = pwndCtrl->GetDlgCtrlID();
        CString str;
        switch (ctrl_ID)
        {     //control ID = EditBox
              case IDC_EditEntry:
                    //Text der EditBox auslesen
                    m_EditEntry.GetWindowText(str);
                    //Wert in die ListControl eintragen
                    m_Schrittliste.SetItemText(nItem, nSubItem, str);
                    //der EditBox den Fokus wieder wegnehmen
                    ::SendDlgItemMessage(m_hWnd,IDC_EditEntry,WM_KILLFOCUS,0,0);
                    //EditBox wieder verstecken
                    m_EditEntry.ShowWindow(SW_HIDE);
                    //Wert im Feld aktualisieren
                    Schrittdauer[nItem] = atoi(str);
                    //von der EditBox überdeckte Items neu zeichnen
                    m_Schrittliste.Update(nItem-1);
                    m_Schrittliste.Update(nItem);
                    break;
              default:
                    break;
     }

}
```

Damit die Listenelemente andere Farben erhalten können, ist folgende Methode nötig. Sie verändert in Abhängigkeit vom Wert des Listenelementes die Hintergrund- und Schriftfarbe des Elements.

```
//Das Grundgerüst der folgenden Funktion stammt von Michael Dunn.
//Dies Funktion dient der Änderung der Hintergrund- und Textfarben der
Tabelle
void CPSTGDlg::OnCustomdrawList(NMHDR* pNMHDR, LRESULT* pResult)
{
      NMLVCUSTOMDRAW* pLVCD = reinterpret_cast<NMLVCUSTOMDRAW*>( pNMHDR );

      //Take the default processing unless we set this to something else
      //below.
      *pResult = 0;

      // First thing - check the draw stage. If it's the control's prepaint
      // stage, then tell Windows we want messages for every item.

      if ( CDDS_PREPAINT == pLVCD->nmcd.dwDrawStage )
      {
            *pResult = CDRF_NOTIFYITEMDRAW;
      }
      else if ( CDDS_ITEMPREPAINT == pLVCD->nmcd.dwDrawStage )
      {
      // This is the notification message for an item.  We'll request
      // notifications before each subitem's prepaint stage.
            *pResult = CDRF_NOTIFYSUBITEMDRAW;
      }
      else if ((CDDS_ITEMPREPAINT | CDDS_SUBITEM) == pLVCD->nmcd.dwDrawStage)
      {
      // This is the prepaint stage for a subitem. Here's where we set the
      // item's text and background colors. Our return value will tell
      // Windows to draw the subitem itself, but it will use the new colors
      // we set here.

            COLORREF clrNewTextColor, clrNewBkColor;
            int    nItem = static_cast<int>( pLVCD->nmcd.dwItemSpec );
```

```
            CString strTemp = m_Schrittliste.GetItemText(  nItem,
                                                           pLVCD->iSubItem);

            if (Kanalwert[nItem] & 16777216) clrNewTextColor = RGB(255,128,0);
            else clrNewTextColor = RGB(0,0,0);

            if(strTemp == "ON")
                  //Set the bkgrnd color to green
                  clrNewBkColor = RGB(0,255,0);
            else
                  //leave the bkgrnd color white
                  clrNewBkColor = RGB(255,255,255);

            // Store the colors back in the NMLVCUSTOMDRAW struct.
            pLVCD->clrText = clrNewTextColor;
            pLVCD->clrTextBk = clrNewBkColor;

            // Tell Windows to paint the control itself.
            *pResult = CDRF_DODEFAULT;
      }
}
```

Änderungen am Rezept, also Hinzufügen und Löschen von Zeilen werden durch folgende Methoden erledigt. Vor jeder Änderung wird der Feldinhalt in die Backup-Felder gesichert.

```
//hängt eine Zeile an das Ende des Rezeptes an
void CPSTGDlg::OnZeileanh()
{
      // Ist kein Rezept geöffnet, passiert nix
      if (maxSchritte == 0) return;

      gespeichert = FALSE;
      m_modified = "modified";
      UpdateData(FALSE);

      // Rezept vor der Veränderung zwischenspeichern
      backup(TRUE);

      char temp[20];
      // Nummer des neuen Schrittes in einen String umwandeln
      itoa(aktSchrittanzahl+1, temp, 10);
      // neue Zeile an das Ende der Liste anhängen
      m_Schrittliste.InsertItem(aktSchrittanzahl, temp);
      // Dauer des neuen Schrittes auf 1 s setzen
      m_Schrittliste.SetItemText(aktSchrittanzahl, 1, "1");
      // Alle Kanäle der neuen Zeile deaktivieren
      for (i=2; i<27; i++)
            m_Schrittliste.SetItemText(aktSchrittanzahl, i, "OFF");
      // neuen Schritt in die Felder eintragen
      Schrittdauer[aktSchrittanzahl] = 1;
      Kanalwert[aktSchrittanzahl] = 0;

      aktSchrittanzahl++;

      // Felder vergrößern, wenn nötig
      if (aktSchrittanzahl + 1 >= maxSchritte)
            if (!Felder_erweitern(10)) return;

      m_Schrittliste.SetFocus();
      VERIFY(m_Schrittliste.EnsureVisible(aktSchrittanzahl-1, FALSE));
}
```

```
//Fügt eine neue Zeile vor der aktuell markierten ein
void CPSTGDlg::OnZeileeinf()
{
	// Ist kein Rezept geöffnet, passiert nix
	if (maxSchritte == 0) return;

	//Position des markierten Elements bestimmen
	POSITION pos = m_Schrittliste.GetFirstSelectedItemPosition ();
	if (pos == NULL) return;
	nItem = m_Schrittliste.GetNextSelectedItem (pos);

	if (nItem > aktSchrittanzahl) return;

	gespeichert = FALSE;
	m_modified = "modified";
	UpdateData(FALSE);

	//vor Änderungen zwischenspeichern
	backup(TRUE);

	char temp[20];

	//Schrittnummer der folgenden Schritte um 1 erhöhen
	for (i=nItem;i<aktSchrittanzahl;i++)
	{
		itoa(i+2, temp, 10);
		VERIFY(m_Schrittliste.SetItemText(i, 0, temp));
	}

	itoa(nItem+1, temp, 10);

	//neue Zeile einfügen vor der aktuell selektierten
	m_Schrittliste.InsertItem(nItem, temp);

	//Schrittdauer = 1
	m_Schrittliste.SetItemText(nItem, 1, "1");

	//alle Kanäle auf "OFF"
	for (i=2; i<27; i++)
		m_Schrittliste.SetItemText(nItem, i, "OFF");

	aktSchrittanzahl++;

	//Felder erweitern wenn nötig
	if (aktSchrittanzahl + 1 >= maxSchritte)
		if (!Felder_erweitern(10)) return;

	//Feldelemente eins nach hinten schieben
	for (i=aktSchrittanzahl; i>nItem; i--)
	{
		Schrittdauer[i] = Schrittdauer[i-1];
		Kanalwert[i] = Kanalwert[i-1];
	}

	Schrittdauer[nItem] = 1;
	Kanalwert[nItem] = 0;

	m_Schrittliste.SetFocus();
}

//entfernt die aktuell markierten Zeilen
void CPSTGDlg::OnZeileentf()
{
	//Ist kein Rezept geöffnet, passiert nix
	if (maxSchritte == 0) return;
```

```
	//Position des markierten Elements bestimmen
	POSITION pos = m_Schrittliste.GetFirstSelectedItemPosition ();

	gespeichert = FALSE;
	m_modified = "modified";
	UpdateData(FALSE);

	//vor Veränderungen zwischenspeichern
	backup(TRUE);

	char temp[20];

	//solange noch markierte Elemente existieren ist pos ungleich Null
	while (pos)
	{
		pos = m_Schrittliste.GetFirstSelectedItemPosition();
		if (pos == NULL) return;

		nItem = m_Schrittliste.GetNextSelectedItem (pos);

		if (nItem > aktSchrittanzahl) return;

		//Schrittnummer der folgenden Schritte um 1 erniedrigen
		for (i=nItem;i<aktSchrittanzahl;i++)
		{
			itoa(i, temp, 10);
			VERIFY(m_Schrittliste.SetItemText(i, 0, temp));
		}

		VERIFY(m_Schrittliste.DeleteItem(nItem));

		//Element aus den Feldern löschen, d. h. die nachfolgenden eins
		//nach vorn schieben
		for (i=nItem; i<aktSchrittanzahl; i++)
		{
			Schrittdauer[i] = Schrittdauer[i+1];
			Kanalwert[i] = Kanalwert[i+1];
		}

		aktSchrittanzahl--;
	}
}
```

Die Felder werden dynamisch an die Größe des Rezeptes angepasst, d. h. sie werden vergrößert, wenn das Rezept mehr Elemente enthalten soll, als die Felder aufnehmen können. Sie werden allerdings nicht wieder verkleinert, wenn Schritte gelöscht werden. Zur Größenänderung werden zwei temporäre Felder angelegt, in die der Inhalt der alten Felder kopiert wird. Danach werden die alten Felder gelöscht, um *elemente* größer wieder neu angelegt und mit dem Inhalt der temporären gefüllt. Letztere werden am Ende wieder gelöscht.

```
//erweitert die Felder zur Speicherung des Rezeptes
int CPSTGDlg::Felder_erweitern(int elemente)
{
	if (elemente > 0)
	{
		int maxneu, i;
		int *temp1 = NULL;
```

```
            ULONG *temp2 = NULL;

            maxneu = maxSchritte + elemente;
            //neues Feld ist um 'elemente' Schritte länger

            //temporäre Felder anlegen
            temp1 = (int *)calloc(maxSchritte,sizeof(int));
            temp2 = (ULONG *)calloc(maxSchritte,sizeof(ULONG));

            if ((temp1 == 0)||(temp2 == 0))
            {
                  MessageBox("Unable to allocate memory for temporary arrays!"
                  ,"Insufficient memory",MB_OK);
                  free(temp1);
                  free(temp2);
                  return 0;
            }

            for (i=0;i<maxSchritte;i++) //Feldinhalt kopieren
            {
                  temp1[i] = Schrittdauer[i];
                  temp2[i] = Kanalwert[i];
            }
            free (Schrittdauer); //alte Felder freigeben
            free (Kanalwert);

            //neues Feld mit neuer Größe anlegen
            Schrittdauer = (int *)calloc(maxneu,sizeof(int));
            Kanalwert = (ULONG *)calloc(maxneu,sizeof(ULONG));

            if ((Schrittdauer == 0)||(Kanalwert == 0))
            {
                  MessageBox("Unable to allocate enlarged arrays for the
                  recipe !","Insufficient memory",MB_OK);
                  free(temp1);
                  free(temp2);
                  return 0;
            }

            for (i=0;i<maxSchritte;i++) //Feldinhalt kopieren
            {
                  Schrittdauer[i] = temp1[i];
                  Kanalwert[i] = temp2[i];
            }

            free (temp1); //temporäre Felder freigeben
            free (temp2);

            //neue Größe für maximale Schrittanzahl übernehmen
            maxSchritte = maxneu;

            return 1;
      }
      return 0;
}
```

Zur Realisierung der Undo-Funktion speichert die folgende Methode den Inhalt der Rezept-Felder oder stellt ihn wieder her. Soll der Inhalt gesichert werden, werden zwei Backup-Felder angelegt, die genauso groß sind wie die aktuellen Felder, und mit den Rezeptdaten gefüllt. Zum Wiederherstellen, das mit dem Undo-Button ausgelöst wird, werden zuerst die aktuellen Felder mit dem Inhalt der Backup-Felder überschrieben.

Danach werden alle Elemente der Schrittliste gelöscht und mit den wiederhergestellten Werten neu gefüllt.

```
//sichert den Inhalt des Rezeptes vor Veränderungen oder stellt die alten
//Daten wieder her
bool CPSTGDlg::backup(bool save)
//save = TRUE -> Speichern, FALSE -> Wiederherstellen
{
	char buffer[20] = "";
	int i, j;

	if (save)
	{
		free(Schrittdauer_bak);
		free(Kanalwert_bak);

		//neue Backup-Felder mit neuer Größe anlegen
		Schrittdauer_bak = (int *)calloc(maxSchritte,sizeof(int));
		Kanalwert_bak = (ULONG *)calloc(maxSchritte,sizeof(ULONG));

		//Feldinhalte kopieren
		memcpy(Schrittdauer_bak, Schrittdauer, maxSchritte * sizeof(int));
		memcpy(Kanalwert_bak, Kanalwert, maxSchritte * sizeof(int));

		Schrittanzahl_bak = aktSchrittanzahl;

		m_Undo.EnableWindow(TRUE); //Undo-Button freischalten
	}
	else
	{
		//Feldinhalte zurückkopieren
		memcpy(Schrittdauer, Schrittdauer_bak, maxSchritte * sizeof(int));
		memcpy(Kanalwert, Kanalwert_bak, maxSchritte * sizeof(int));

		aktSchrittanzahl = Schrittanzahl_bak;

		m_Undo.EnableWindow(FALSE); //Undo-Button sperren

		m_Schrittliste.DeleteAllItems(); //Liste löschen

		//Feldinhalt in die Schrittliste eintragen
		for (i = 0; i<aktSchrittanzahl; i++)
		{
			//Schrittnummer eintragen
			_itoa(i+1, buffer, 10);
			m_Schrittliste.InsertItem(i,buffer);

			//Schrittdauer eintragen
			_itoa(Schrittdauer[i], buffer, 10);
			m_Schrittliste.SetItemText(i,1, buffer);

			//Kanalwerte eintragen
			for (j = 0; j<25; j++)
			{
				if (Kanalwert[i] & (ULONG)pow(2, j))
					m_Schrittliste.SetItemText(i,j + 2, "ON");
				else m_Schrittliste.SetItemText(i,j + 2, "OFF");
			}
		}
	}
	return TRUE;
}

//ruft backup auf zur Wiederherstellung der alten Daten auf
```

```
void CPSTGDlg::OnUndo()
{
	backup(FALSE);
}
```

Die Methoden zum Kopieren und Einfügen von Schritten arbeiten ebenfalls mit zwei temporären Feldern, *Schrittliste_buffer* und *Kanalwert_buffer*. Diese sind immer nur so groß, dass alle ausgewählten Elemente hineinpassen. Die Kopieren-Methode kopiert nacheinander alle ausgewählten Schritte in die Buffer-Felder. Wenn sich zwischen diesen Elementen auch nicht markierte befinden, entsteht trotzdem aus den markierten ein einziger zusammenhängender Block im Buffer. Dieser Block wird beim Einfügen komplett vor der markierten Zeile eingefügt. Die Buffer-Felder werden danach nicht gelöscht, sodass der Block mehrere Male eingefügt werden kann.

```
//kopiert die markierten Elemente in eine Zwischenablage
void CPSTGDlg::OnKopieren()
{
	//Ist kein Rezept geöffnet, nix machen
	if (maxSchritte == 0) return;

	//Index der ersten ausgewählten Zeile holen
	POSITION pos = m_Schrittliste.GetFirstSelectedItemPosition ();
	if (pos == NULL) return;

	nItem = m_Schrittliste.GetNextSelectedItem (pos);

	if (nItem > aktSchrittanzahl) return;

	bufferlaenge = m_Schrittliste.GetSelectedCount();
	int i = 0;

	free(Schrittdauer_buffer);
	free(Kanalwert_buffer);

	Schrittdauer_buffer = (int *)calloc(bufferlaenge,sizeof(int));
	Kanalwert_buffer = (ULONG *)calloc(bufferlaenge,sizeof(ULONG));

	//alle ausgewählten Elemente in die Buffer kopieren
	while (pos)
	{
		Schrittdauer_buffer[i] = Schrittdauer[nItem];
		Kanalwert_buffer[i] = Kanalwert[nItem];

		i++;

		nItem = m_Schrittliste.GetNextSelectedItem(pos);
	}

	Schrittdauer_buffer[i] = Schrittdauer[nItem];
	Kanalwert_buffer[i] = Kanalwert[nItem];

	m_Schrittliste.SetFocus();
}

//fügt die Elemente der Zwischenablage vor der aktuell markierten Zeile ein
void CPSTGDlg::OnEinfuegen()
{
```

```
	//Ist kein Rezept geöffnet, nix machen
	if (maxSchritte == 0) return;

	//Index der ausgewählten Zeile holen
	POSITION pos = m_Schrittliste.GetFirstSelectedItemPosition ();
	if (pos == NULL) return;
	nItem = m_Schrittliste.GetNextSelectedItem (pos);

	if (nItem > aktSchrittanzahl) return;

	gespeichert = FALSE;
	m_modified = "modified";
	UpdateData(FALSE);

	// vorher sichern
	backup(TRUE);

	char temp[20];
	int i, j, k;

	for (k=bufferlaenge - 1; k>-1; k--)
	{

		//Schrittnummer der folgenden Schritte um 1 erhöhen
		for (i=nItem;i<aktSchrittanzahl;i++)
		{
			itoa(i+2, temp, 10);
			VERIFY(m_Schrittliste.SetItemText(i,0,temp));
		}

		//neue Zeile einfügen vor der aktuell selektierten
		itoa(nItem+1, temp, 10);
		m_Schrittliste.InsertItem(nItem, temp);

		aktSchrittanzahl++;

		if (aktSchrittanzahl + 1 >= maxSchritte)
			if (!Felder_erweitern(10)) return;

		//Feldelemente eins nach hinten schieben
		for (i=aktSchrittanzahl; i>nItem; i--)
		{
			Schrittdauer[i] = Schrittdauer[i-1];
			Kanalwert[i] = Kanalwert[i-1];
		}

		Schrittdauer[nItem] = Schrittdauer_buffer[k];
		Kanalwert[nItem] = Kanalwert_buffer[k];

		//Schrittdauer eintragen
		_itoa(Schrittdauer_buffer[k], temp, 10);
		m_Schrittliste.SetItemText(nItem,1, temp);

		for (j = 0; j<25; j++) //Kanalwerte eintragen
		{
			if (Kanalwert_buffer[k] & (ULONG)pow(2, j))
				m_Schrittliste.SetItemText(nItem,j + 2, "ON");
			else  m_Schrittliste.SetItemText(nItem,j + 2, "OFF");
		}
	}
	m_Schrittliste.SetFocus();
	VERIFY(m_Schrittliste.SetItemState(nItem, LVIS_SELECTED,LVIS_SELECTED));
}
```

6.2 Quelltext des Ausführungsprogramms

Es werden hier wieder zwei Felder zur Speicherung des Rezeptes verwendet, *Schrittdauer* und *Relaiswert*. Diese werden beim Öffnen eines Rezeptes mit Werten gefüllt. Jedes Bit von *Relaiswert* entspricht dem Zustand eines Ausgangsrelais.

Nach dem Starten des Programms wird zuerst die Konfigurationsdatei *Config.ini* eingelesen. Editor und Ausführungsprogramm benutzen dabei die gleiche Datei.

Die Ausführung eines Rezeptes wird mit Hilfe eines Windows-Timers gesteuert. Dieser wird so eingestellt, dass im Abstand von einer Sekunde die Funktion *OnTimer* aufgerufen wird, die durch das Rezept schreitet. Die festgelegten Ausgänge werden dann von der Funktion *Ausgeben* an die Schaltung übertragen. Die Variable *prozlaeuft* bestimmt, ob das Rezept weiter abgearbeitet wird oder nicht.

Das Datei-Einlesen und das Felder-Erweitern funktioniert wie im Editorprogramm.

Es folgt nun eine Übersicht über die deklarierten Variablen und Methoden, entnommen aus der Headerdatei PXDlg.h:

```
int Felder_erweitern(int elemente);
HICON m_hIcon;
CString vollstDateiname, Dateiname, Pfad;
CStdioFile Konfdatei, PAPdatei, Debugdatei;
CFileException error;
CString Bezeichner[25];
CString Fehler, FehlerTitel;
int *Schrittdauer;
ULONG *Relaiswert;
int maxSchritte, Schrittanzahl, aktSchritt;
int i;
bool prozlaeuft;
char ComPort[5];
HANDLE hPort;
DCB dcb;
UINT tnummer;
int daten1[18], daten2[18], daten3[18];
int aktDL, aktLaufzeit, Blockanfang, Gesamtdurchlauf, aktGDL, GesamtLaufzeit;
int Startschritt;
bool debugenable;
bool bExec;
CString cmdline, cmdlinedateiname;
int startphase;
COLORREF laeuft, pause, abbruch;

bool CPXDlg::KonfigurationLesen();
bool CPXDlg::DateiEinlesen();
void CPXDlg::Ausgeben();
```

Einige der Variablen werden in der Implementierungsdatei PXDlg.cpp in der Methode PXDLG::INITDIALOG wie folgt initialisiert:

```
maxSchritte = 50;
aktSchritt = 0;
Schrittanzahl = 0;
Gesamtdurchlauf = 1;
tnummer = 0;
debugenable = FALSE;
Schrittdauer = NULL;
Relaiswert = NULL;
i = 0;
hPort = NULL;
aktDL = 0;
aktLaufzeit = 0;
Blockanfang = 0;
aktGDL = 0;
GesamtLaufzeit = 0;
Startschritt = 0;
prozlaeuft = FALSE;
startphase = 0;

//Schriftgröße für die Status- und Ausgangszeile erhöhen
CFont font;
VERIFY(font.CreatePointFont(140, "Arial", NULL));
m_CStatus.SetFont(&font, TRUE);
m_CRelaiswert.SetFont(&font, TRUE);

//Hintergrundfarben für die Statuszeile festlegen
laeuft = RGB(0,255,0);
pause = RGB(254,234,131);
abbruch = RGB(255,0,0);

//Datenfelder allokieren
Schrittdauer = (int *)calloc(maxSchritte,sizeof(int));
Relaiswert = (ULONG *)calloc(maxSchritte,sizeof(ULONG));

if ((Schrittdauer == 0)||(Relaiswert == 0))
{
      Fehler.LoadString(IDS_Speicherfehler);
      MessageBox(Fehler,"Error",MB_OK);
      OnOK();
}

ComPort[0] = 'C'; //Standard-Schnittstelle
ComPort[1] = 'O';
ComPort[2] = 'M';
ComPort[3] = '1';
ComPort[4] = '\0';


//Daten initialisieren
daten1[0]  = 0;
daten1[1]  = 1;
daten1[2]  = 0;
daten1[3]  = 0;
daten1[4]  = 0; // A2
daten1[5]  = 0; // A1
daten1[6]  = 0; // A0
daten1[7]  = 0;
daten1[8]  = 1; //ACK
daten1[9]  = 0; //Bit7
daten1[10] = 0; //Bit6
daten1[11] = 0; //Bit5
daten1[12] = 0; //Bit4
daten1[13] = 0; //Bit3
daten1[14] = 0; //Bit2
```

```
daten1[15] = 0; //Bit1
daten1[16] = 0; //Bit0
daten1[17] = 1; //ACK

daten2[0]  = 0;
daten2[1]  = 1;
daten2[2]  = 0;
daten2[3]  = 0;
daten2[4]  = 0; // A2
daten2[5]  = 0; // A1
daten2[6]  = 1; // A0
daten2[7]  = 0;
daten2[8]  = 1; //ACK
daten2[9]  = 0; //Bit7
daten2[10] = 0; //Bit6
daten2[11] = 0; //Bit5
daten2[12] = 0; //Bit4
daten2[13] = 0; //Bit3
daten2[14] = 0; //Bit2
daten2[15] = 0; //Bit1
daten2[16] = 0; //Bit0
daten2[17] = 1; //ACK

daten3[0]  = 0;
daten3[1]  = 1;
daten3[2]  = 0;
daten3[3]  = 0;
daten3[4]  = 0; // A2
daten3[5]  = 1; // A1
daten3[6]  = 0; // A0
daten3[7]  = 0;
daten3[8]  = 1; //ACK
daten3[9]  = 0; //Bit7
daten3[10] = 0; //Bit6
daten3[11] = 0; //Bit5
daten3[12] = 0; //Bit4
daten3[13] = 0; //Bit3
daten3[14] = 0; //Bit2
daten3[15] = 0; //Bit1
daten3[16] = 0; //Bit0
daten3[17] = 1; //ACK
```

Die Abfrage der Kommandozeilenoptionen erfolgt in der gleichen Methode im folgenden Codefragment:

```
cmdline = ::GetCommandLine();
cmdlinedateiname = "";
bExec = FALSE;

if(!cmdline.IsEmpty()) //Kommandozeilenoptionen einlesen
{
      int danfang, dende;
      danfang = dende = -1;
      cmdline.MakeUpper();
      danfang = cmdline.Find("-FILE:",0); //Beginn des Dateinamenparameters
      dende = cmdline.Find(".RCP",0); //Ende des Dateinamens
      if ((dende > danfang + 6)&&(danfang > 0))
            cmdlinedateiname = cmdline.Mid(danfang + 6, dende - danfang - 2);
            //Dateiname kopieren
      else cmdlinedateiname = "";
      if ((cmdline.Find("-EXEC",0) > dende + 4)&&(!cmdlinedateiname.IsEmpty
()))
            bExec = TRUE; //Execparameter an gültiger Position
```

```
}
if (!cmdlinedateiname.IsEmpty() && hPort) OnOeffnen();
```

Die Behandlungsroutinen für die Buttons sind im folgenden Code zu sehen. Die EnableWindow-Methode der Buttons sperrt (bei FALSE als Übergabewert) bzw. schaltet den entsprechenden Button wieder frei (bei TRUE als Übergabewert).

```
//Klick auf den Exit-Button
void CPXDlg::OnOK()
{
	free(Schrittdauer);
	free(Relaiswert);
	if (hPort) CloseHandle(hPort);
	CDialog::OnOK();
}

//Klick auf den Open-Button
void CPXDlg::OnOeffnen()
{
	m_CStatus.SetBkColor(CLR_DEFAULT);
	if (cmdlinedateiname == "")
	{
		CFileDialog DateiDialog(
			TRUE,  // TRUE = Datei oeffnen
			"rcp",  // Standardwert für anzuhängende Dateierweiterung
			NULL,  // Standardwert für das Feld des Dateinamen
			NULL,  // OPENFILENAME-Flags
			"Recipe file (*.rcp)|*.rcp|all files|*.*|", // Filterregeln
			NULL); // Verweis auf Elterndialog
		DateiDialog.DoModal(); //Dialog öffnen
		vollstDateiname = DateiDialog.GetPathName();
		Dateiname = DateiDialog.GetFileName();
		Pfad = DateiDialog.GetPathName();
	}
	else
	{
		vollstDateiname = cmdlinedateiname;
		cmdlinedateiname = "";
	}
	m_EditDateiname = vollstDateiname;
	//Eingabefeld den kompletten Pfad der Datei übergeben
	if (!vollstDateiname.IsEmpty())
		if (!DateiEinlesen())
		{
			m_Status = "no file opened";
			m_BnOeffnen.EnableWindow(TRUE);
			m_BnOK.EnableWindow(TRUE);
			m_BnStart.EnableWindow(FALSE);
			m_BnAbbrechen.EnableWindow(FALSE);
			m_BnPause.EnableWindow(FALSE);
		}
		else
		{
			m_Status = Dateiname + " opened";
			m_BnStart.EnableWindow(TRUE);
			m_BnOK.EnableWindow(TRUE);
			m_BnOeffnen.EnableWindow(TRUE);
			m_BnAbbrechen.EnableWindow(FALSE);
			m_BnPause.EnableWindow(FALSE);
			if (bExec) OnStart();
		}
	UpdateData(FALSE);  //Daten im Dialog aktualisieren
}
```

```
//Klick auf den Start-Button, Rezeptabarbeitung anstoßen
void CPXDlg::OnStart()
{
	if (bExec) bExec = FALSE;
	prozlaeuft = TRUE;
	startphase = 0;
	aktGDL = aktDL = aktLaufzeit = aktSchritt = GesamtLaufzeit = 0;

	while (Relaiswert[aktSchritt] & 16777216)
	{	//falls anfangs Wiederholungsschritte stehen, diese überspringen
		aktSchritt++; //nächster Schritt
		if (aktSchritt == Schrittanzahl) //Feldende erreicht
		{
			prozlaeuft = FALSE;

			m_Status = "recipe completed";
			m_BnAbbrechen.EnableWindow(FALSE);
			m_BnPause.EnableWindow(FALSE);
			m_BnStart.EnableWindow(TRUE);
			m_BnOK.EnableWindow(TRUE);
			m_BnOeffnen.EnableWindow(TRUE);
			m_CStatus.SetBkColor(CLR_DEFAULT);
			UpdateData(FALSE);
		}
	}

	if (prozlaeuft)
	{
		//Startschritt gefunden
		Startschritt = Blockanfang = aktSchritt;
		Ausgeben();

		if (tnummer) KillTimer(tnummer);
		tnummer = SetTimer(250, 1000, NULL); //Timer starten

		m_Status = "Initialization phase! Please wait 5 seconds!";
		m_BnAbbrechen.EnableWindow(TRUE);
		m_BnStart.EnableWindow(FALSE);
		m_BnOK.EnableWindow(FALSE);
		m_BnOeffnen.EnableWindow(FALSE);
		m_CStatus.SetBkColor(laeuft);
		UpdateData(FALSE);
	}
}

//Pausieren der Abarbeitung
void CPXDlg::OnPause()
{
	if (prozlaeuft)
	{
		prozlaeuft = FALSE;

		//wenn der Prozess nicht läuft, wird die 0 vom Feldende ausgegeben
		Ausgeben();

		m_Status = "recipe paused";
		m_BnAbbrechen.EnableWindow(TRUE);
		m_BnPause.EnableWindow(TRUE);
		m_BnStart.EnableWindow(FALSE);
		m_BnOK.EnableWindow(FALSE);
		m_BnOeffnen.EnableWindow(FALSE);
		m_CStatus.SetBkColor(pause);
	}
	else
	{
```

```
            prozlaeuft = TRUE;

            Ausgeben();

            m_Status = "recipe running";
            m_BnAbbrechen.EnableWindow(TRUE);
            m_BnPause.EnableWindow(TRUE);
            m_BnStart.EnableWindow(FALSE);
            m_BnOK.EnableWindow(FALSE);
            m_BnOeffnen.EnableWindow(FALSE);
            m_CStatus.SetBkColor(laeuft);
      }
      UpdateData(FALSE);
}

//Rezept abbrechen
void CPXDlg::OnAbbrechen()
{
      prozlaeuft = FALSE;
      aktDL = aktLaufzeit = 0;
      if (tnummer) KillTimer(tnummer);

      //am Ende des Programms folgt in jedem Fall ein Schritt mit 0 als
      //Ausgangswert, dieser wird ausgegeben, wenn der Prozess nicht läuft
      Ausgeben();

      aktSchritt = 0; //aktuellen Schritt wieder auf den Feldanfang

      m_Status = "recipe canceled";
      m_BnStart.EnableWindow(TRUE);
      m_BnOK.EnableWindow(TRUE);
      m_BnOeffnen.EnableWindow(TRUE);
      m_BnAbbrechen.EnableWindow(FALSE);
      m_BnPause.EnableWindow(FALSE);
      m_CStatus.SetBkColor(abbruch);
      UpdateData(FALSE);
}
```

Die Übertragung der aktuellen Zustände der Relais erfolgt in folgender Methode:

```
//aktuellen Zustand der Ausgänge an die Schaltung übertragen
void CPXDlg::Ausgeben()
{
      int j; //Feldelementindex, der auf das auszugebende Feldelement zeigt
      if (prozlaeuft)
            j = aktSchritt; //aktuelles Feldelement wird ausgegeben
      else
            j = Schrittanzahl; //die 0 am Feldende wird ausgegeben

      //Daten für die Übertragung eintragen
      daten1[9]  = Relaiswert[j] & 128;     //Bit7
      daten1[10] = Relaiswert[j] & 64;      //Bit6
      daten1[11] = Relaiswert[j] & 32;      //Bit5
      daten1[12] = Relaiswert[j] & 16;      //Bit4
      daten1[13] = Relaiswert[j] & 8;       //Bit3
      daten1[14] = Relaiswert[j] & 4;       //Bit2
      daten1[15] = Relaiswert[j] & 2;       //Bit1
      daten1[16] = Relaiswert[j] & 1;       //Bit0

      daten2[9]  = Relaiswert[j] & 32768;   //Bit7
      daten2[10] = Relaiswert[j] & 16384;   //Bit6
      daten2[11] = Relaiswert[j] & 8192;    //Bit5
      daten2[12] = Relaiswert[j] & 4096;    //Bit4
      daten2[13] = Relaiswert[j] & 2048;    //Bit3
```

```
daten2[14] = Relaiswert[j] & 1024;    //Bit2
daten2[15] = Relaiswert[j] & 512;     //Bit1
daten2[16] = Relaiswert[j] & 256;     //Bit0

daten3[9]  = Relaiswert[j] & 8388608; //Bit7
daten3[10] = Relaiswert[j] & 4194304; //Bit6
daten3[11] = Relaiswert[j] & 2097152; //Bit5
daten3[12] = Relaiswert[j] & 1048576; //Bit4
daten3[13] = Relaiswert[j] & 524288;  //Bit3
daten3[14] = Relaiswert[j] & 262144;  //Bit2
daten3[15] = Relaiswert[j] & 131072;  //Bit1
daten3[16] = Relaiswert[j] & 65536;   //Bit0

if (hPort)
{
      //erstes Datenpaket ausgeben
      //Stop-Condition
      EscapeCommFunction(hPort, SETRTS); //SCL auf 1 etwa 100µs
      EscapeCommFunction(hPort, SETDTR); //SDA auf 1 etwa 27µs

      //Start-Condition:
      EscapeCommFunction(hPort, CLRDTR); //SDA auf 0 etwa 40µs
      EscapeCommFunction(hPort, CLRRTS); //SCL auf 0 etwa 27µs

      for (i=0; i<18; i++)
      {
            if (daten1[i]) EscapeCommFunction(hPort,SETDTR); //SDA auf 1
            else EscapeCommFunction(hPort, CLRDTR); //SDA auf 0
            EscapeCommFunction(hPort, SETRTS); //SCL auf 1
            EscapeCommFunction(hPort, CLRRTS); //SCL auf 0
      }

      EscapeCommFunction(hPort, CLRDTR); //SDA auf 0

      //Stop-Condition:
      EscapeCommFunction(hPort, SETRTS); //SCL auf 1
      EscapeCommFunction(hPort, SETDTR); //SDA auf 1

      //zweites Datenpaket ausgeben
      //Start-Condition:
      EscapeCommFunction(hPort, CLRDTR); //SDA auf 0 etwa 40µs
      EscapeCommFunction(hPort, CLRRTS); //SCL auf 0 etwa 27µs

      for (i=0; i<18; i++)
      {
            if (daten2[i]) EscapeCommFunction(hPort,SETDTR); //SDA auf 1
            else EscapeCommFunction(hPort, CLRDTR); //SDA auf 0
            EscapeCommFunction(hPort, SETRTS); //SCL auf 1
            EscapeCommFunction(hPort, CLRRTS); //SCL auf 0
      }

      EscapeCommFunction(hPort, CLRDTR); //SDA auf 0

      //Stop-Condition:
      EscapeCommFunction(hPort, SETRTS); //SCL auf 1
      EscapeCommFunction(hPort, SETDTR); //SDA auf 1

      //drittes Datenpaket ausgeben
      //Start-Condition:
      EscapeCommFunction(hPort, CLRDTR); //SDA auf 0 etwa 40µs
      EscapeCommFunction(hPort, CLRRTS); //SCL auf 0 etwa 27µs

      for (i=0; i<18; i++)
      {
            if (daten3[i]) EscapeCommFunction(hPort,SETDTR); //SDA auf 1
            else EscapeCommFunction(hPort, CLRDTR); //SDA auf 0
```

```
                    EscapeCommFunction(hPort, SETRTS); //SCL auf 1
                    EscapeCommFunction(hPort, CLRRTS); //SCL auf 0
                }

                EscapeCommFunction(hPort, CLRDTR); //SDA auf 0

                //Stop-Condition:
                EscapeCommFunction(hPort, SETRTS); //SCL auf 1
                EscapeCommFunction(hPort, SETDTR); //SDA auf 1
        }
}
```

6.3 Schalt- und Verdrahtungspläne

Es folgt zuerst der Schaltplan des Abwärtszählers. Mit den DIP-Schaltern wird der Zählerstand eingestellt und mit dem 'PRELOAD'-Taster geladen. 'MASTERRESET' dient zum Rücksetzen auf Null. Die beiden Fotografien (Bilder 6 und 7) zeigen das fertige Modul im Betrieb und die Schalttafel mit Tymgard und angebrachtem Modul.

Danach ist der Schaltplan der neuen Prozesssteuereinheit abgedruckt (Bild 8). Der Steckverbinder X25 ist die 9-polige Sub-D-Buchse, über die die Steuersignale vom Rechner geliefert werden, und X26 ist der 4-polige Stromversorgungsanschluss, der die für die Schaltung nötige Betriebsspannung, ebenfalls vom Rechner, liefert (siehe Bild 10). Die Anschlüsse X27 und X28 liefern die I2C-Bussignale und die Versorgungsspannungen zur möglichen Erweiterung der Einheit um weitere Ausgänge. Bild 9 zeigt das einseitige Platinenlayout der Einheit in der Ansicht auf die Leitbahnseite der Platine. Im hier dargestellten negativen Format wurde es für die Herstellung verwendet.

Danach folgen die Verdrahtungspläne der Schalttafel nach dem Einbau der Prozesssteuereinheit (Bilder 11, 12 und 13). Die Nummerierung der Schalter ist in der Form „Zeile/Spalte“ gewählt. Damit das Ventil F1 automatisch von der Prozesssteuerung kontrolliert werden kann, wurde anstelle des alten ein neuer Schalter eingebaut und entsprechend verdrahtet (Bild 14). Die nachfolgenden Fotografien (Bilder 15 und 16) zeigen die Front der Schalttafel und die Rückseite mit eingebauter Prozesssteuereinheit.

Dann sind zwei Tabellen abgedruckt. Sie geben Aufschluss über die Belegung der Steckverbinder zwischen Schalttafel und Steuereinheit.

Zuletzt sind die technischen Daten sowohl des Abwärtszählers als auch der neuen Prozesssteuereinheit gegeben.

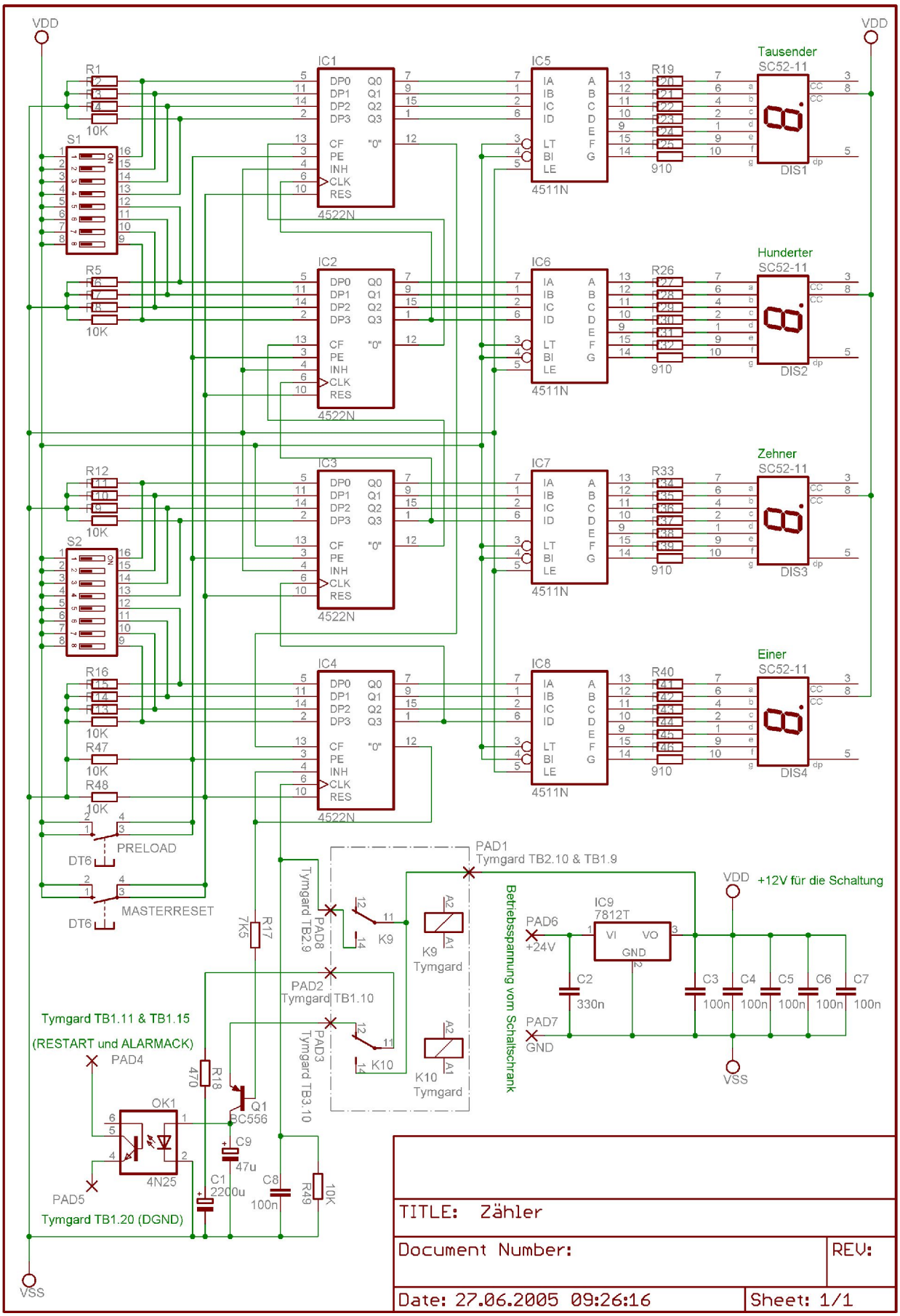

Bild 5: Schaltplan des Abwärtszählers

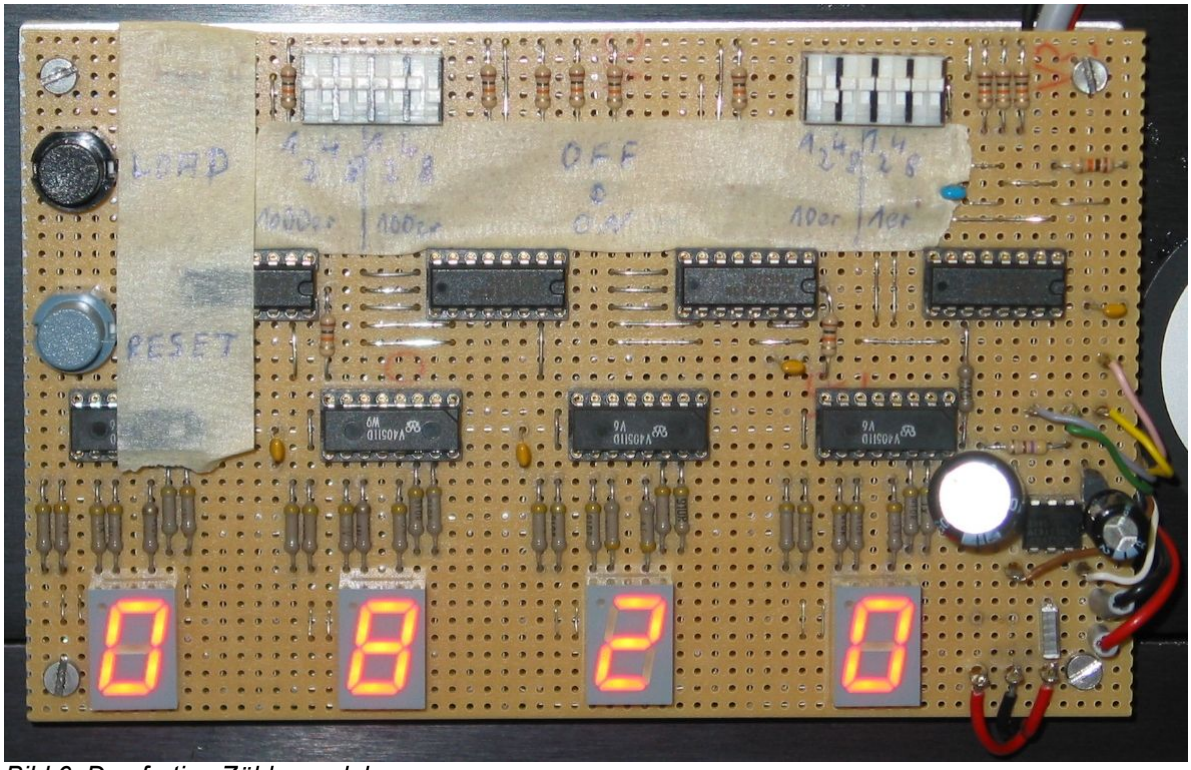

Bild 6: Das fertige Zählermodul

Bild 7: Die Front der Schalttafel mit Zählermodul

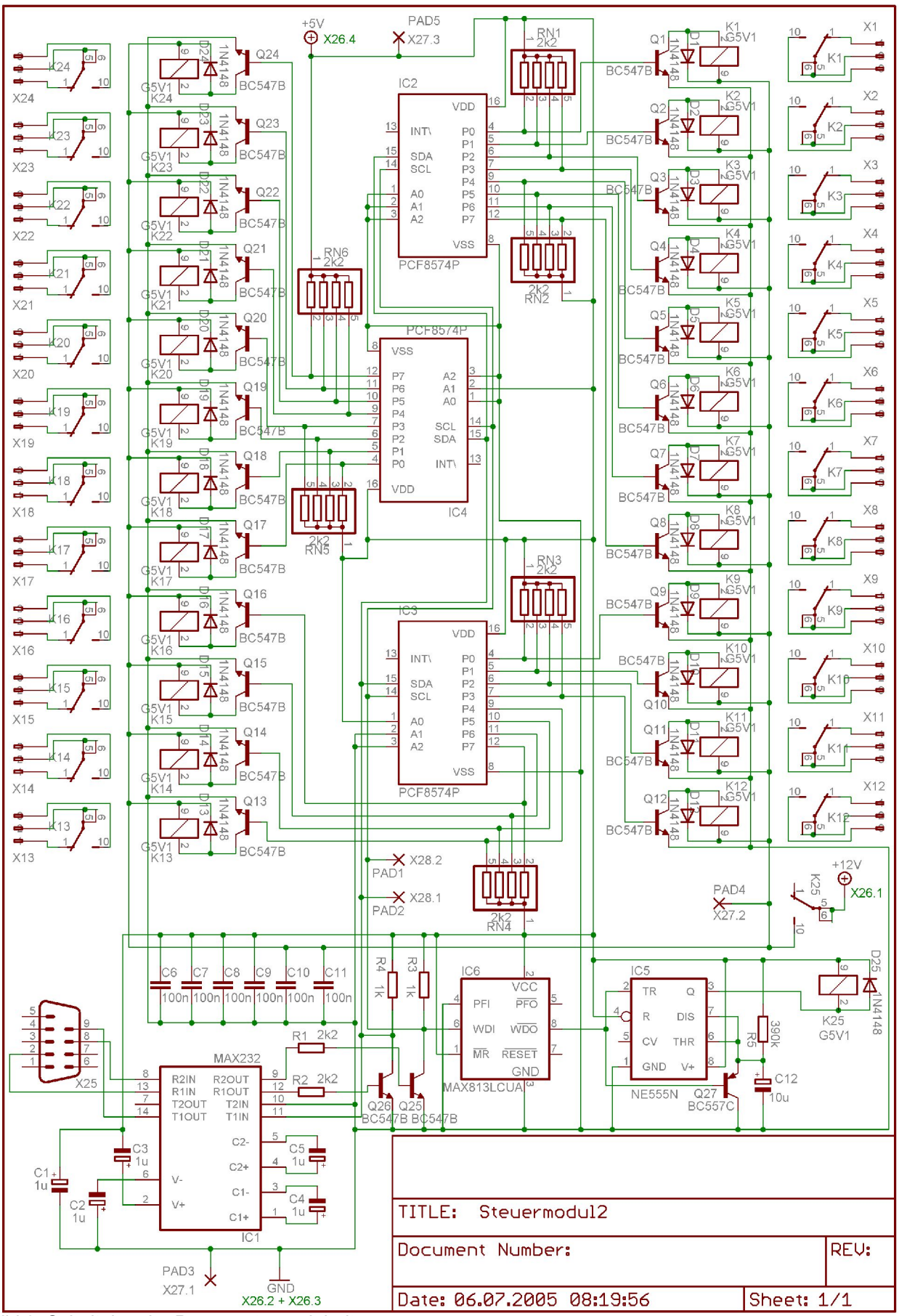

Bild 8: Schaltplan der Prozesssteuereinheit

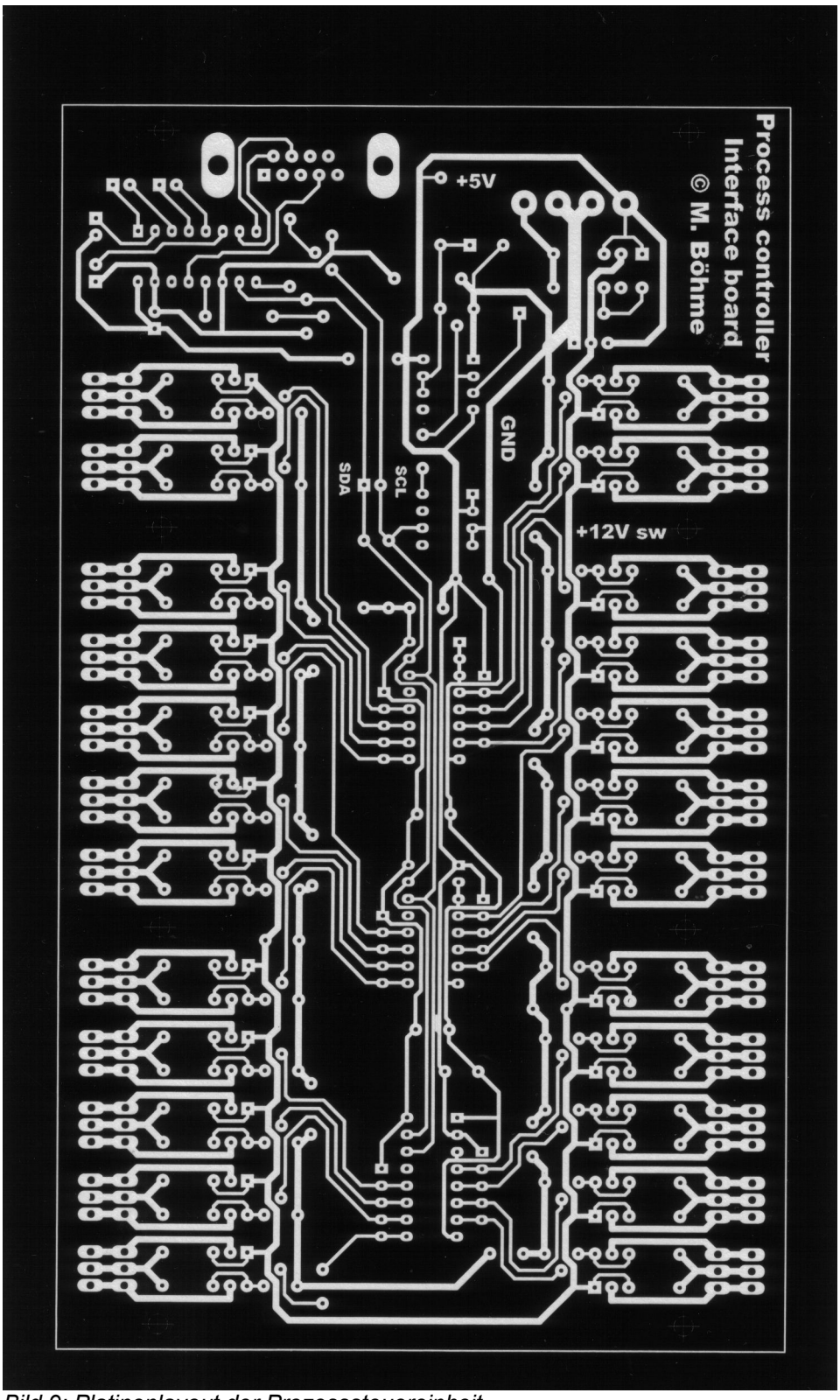

Bild 9: Platinenlayout der Prozesssteuereinheit

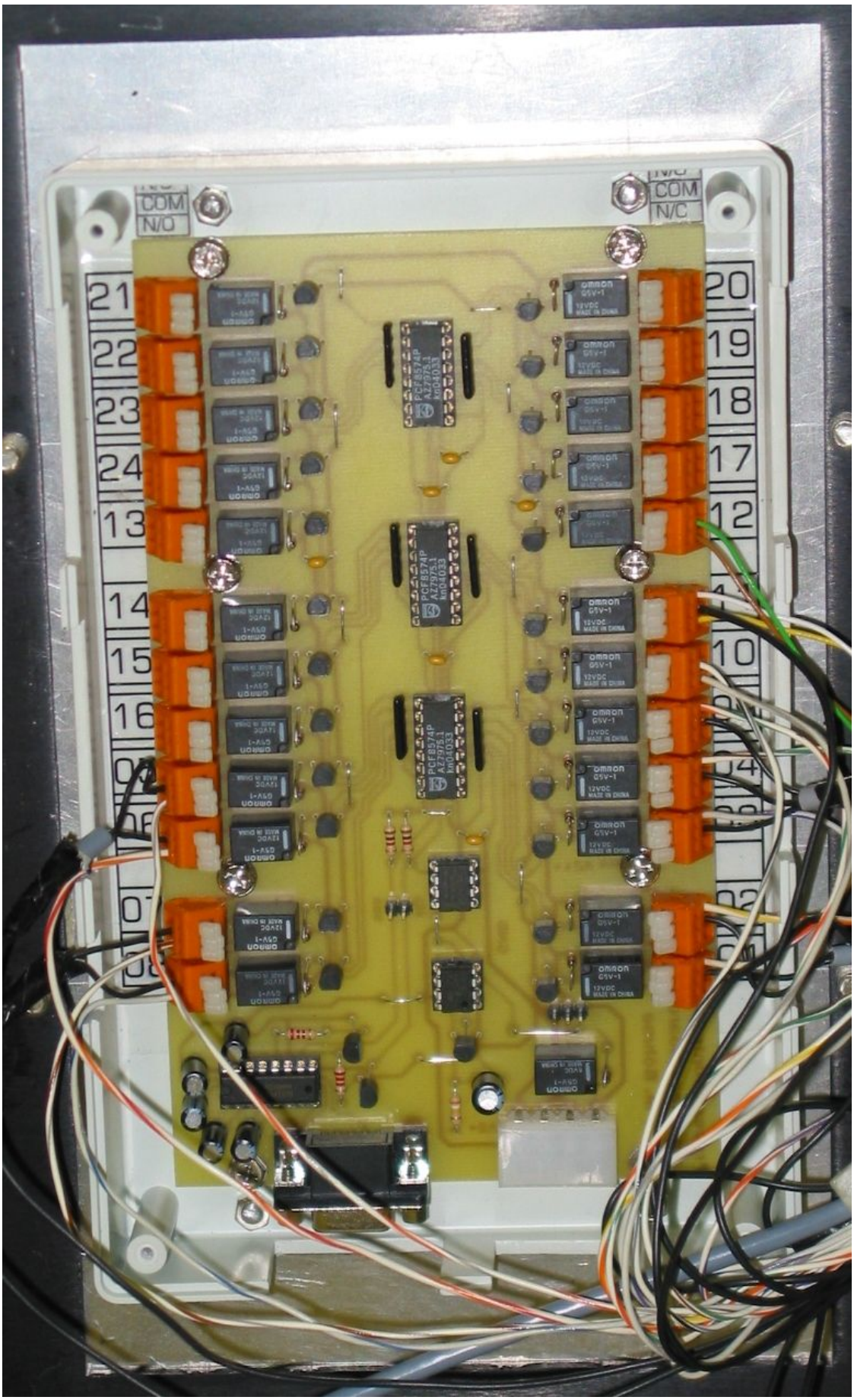

Bild 10: fertig aufgebaute Prozesssteuereinheit

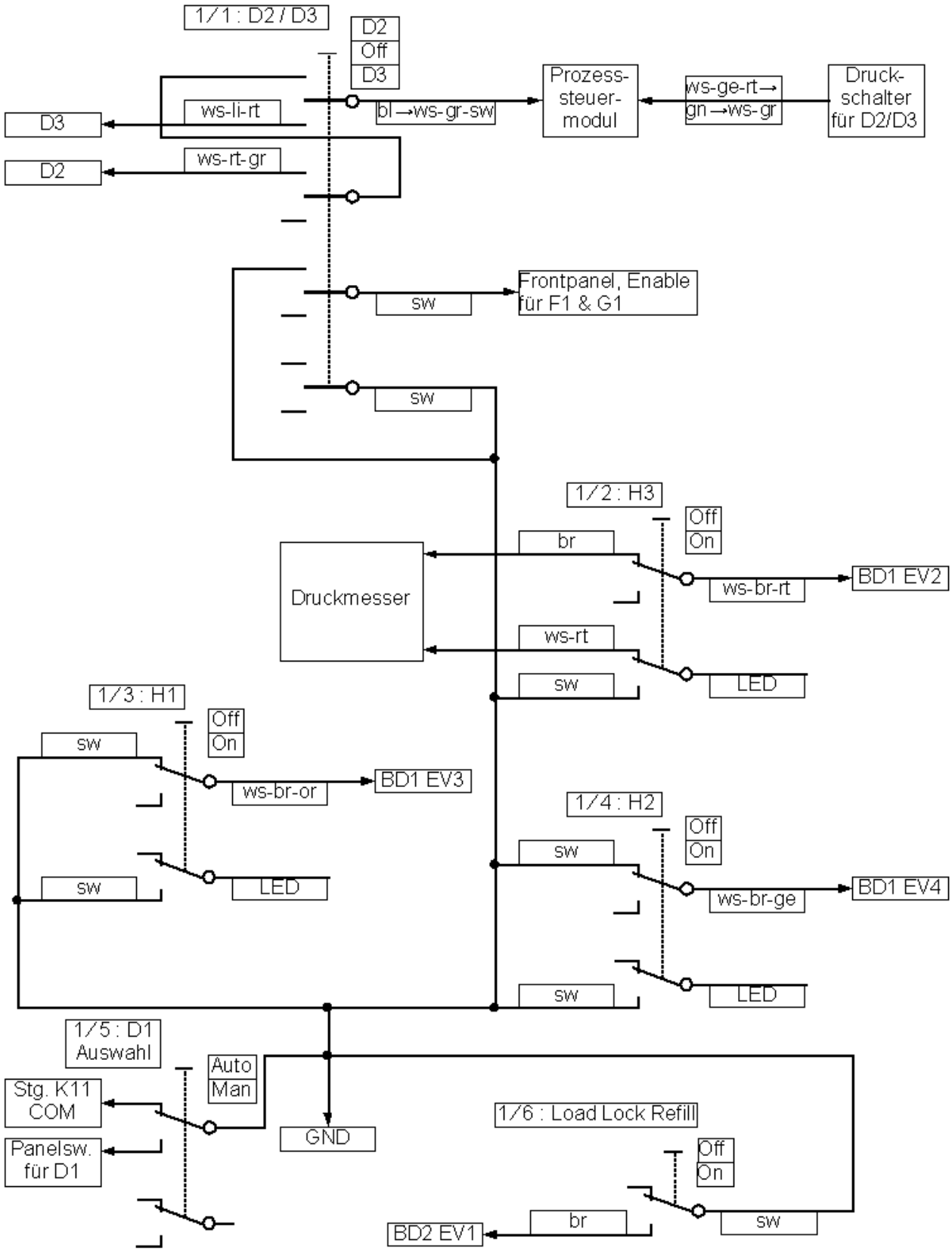

Bild 11: Verdrahtungsplan Seite 1

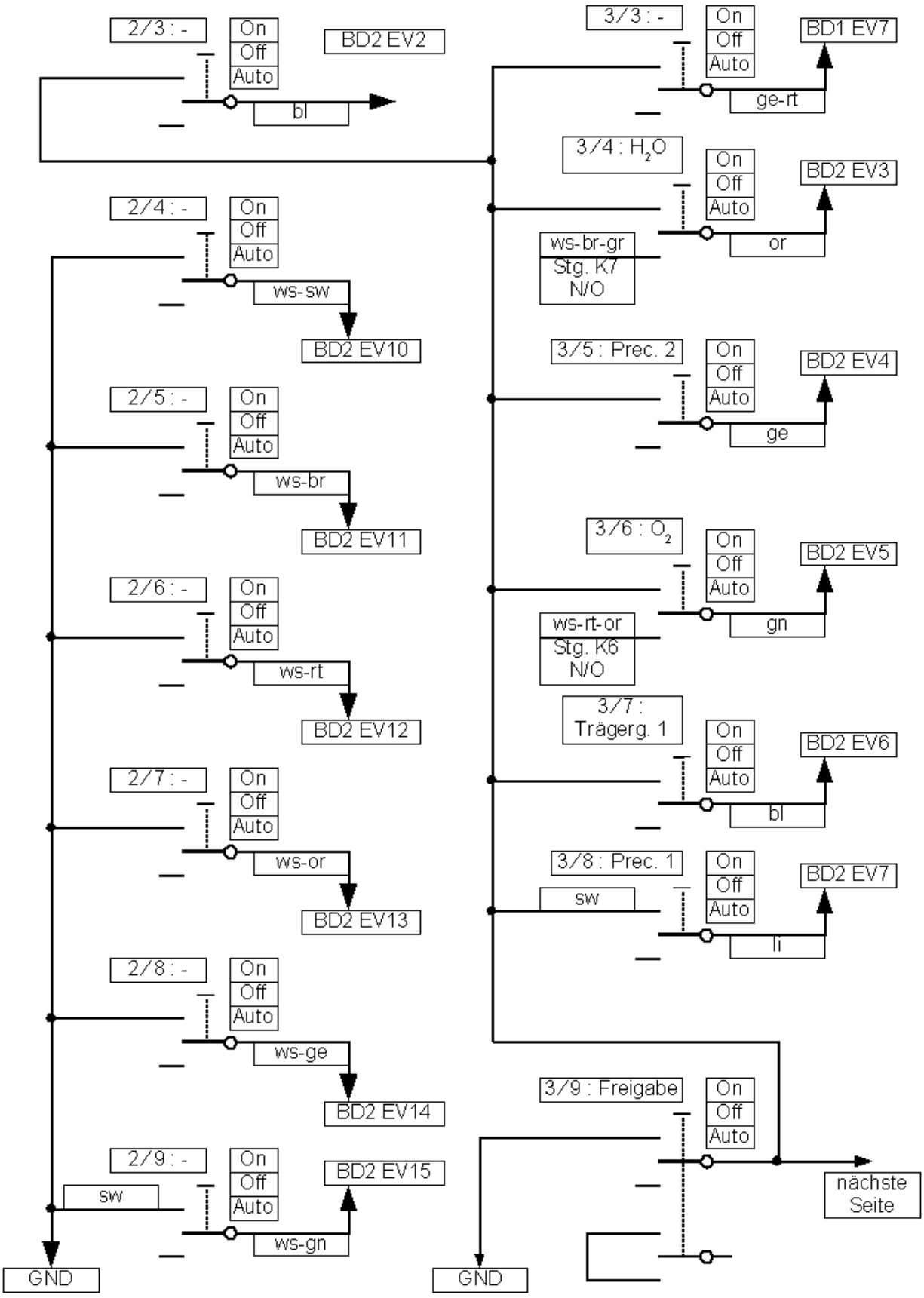

Bild 12:Verdrahtungsplan Seite 2

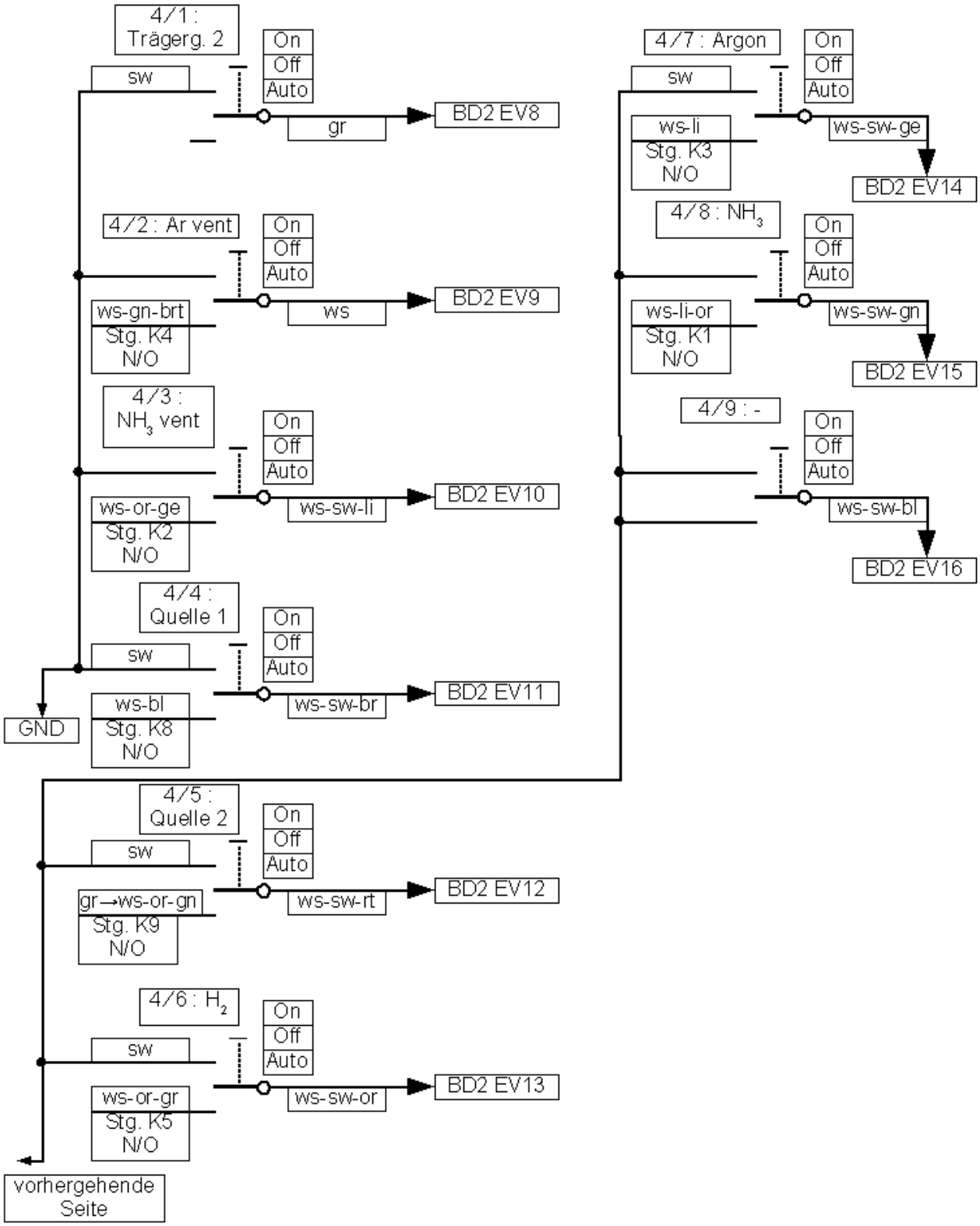

Bild 13:Verdrahtungsplan Seite 3

Die neue Verdrahtung des Schalters zur Steuerung von F1 zeigt das folgende Bild.

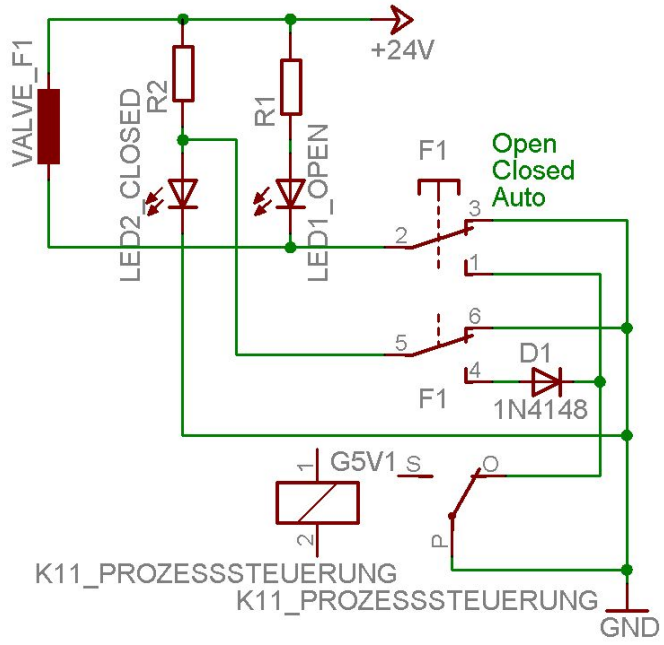

Bild 14: Schalter für F1

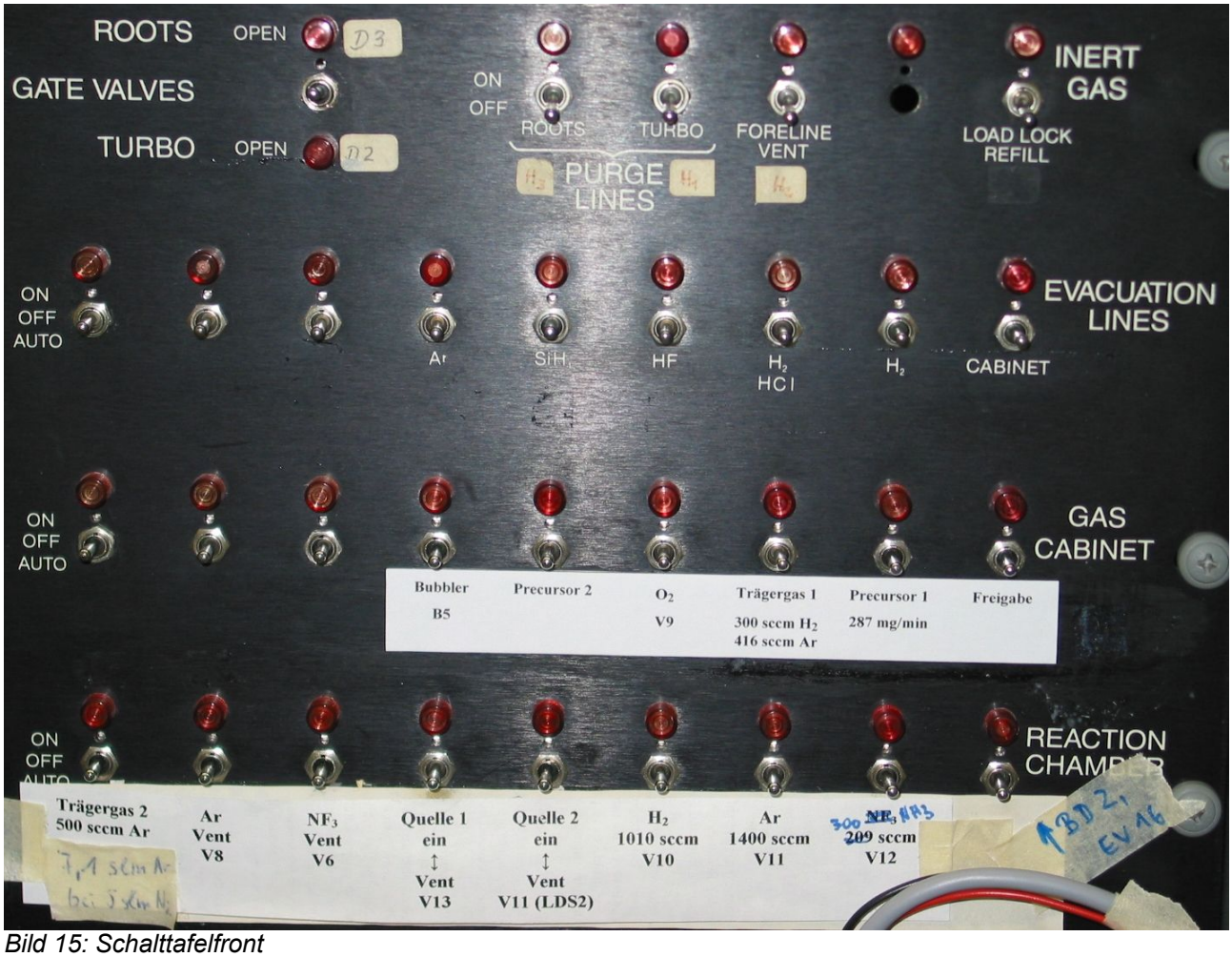

Bild 15: Schalttafelfront

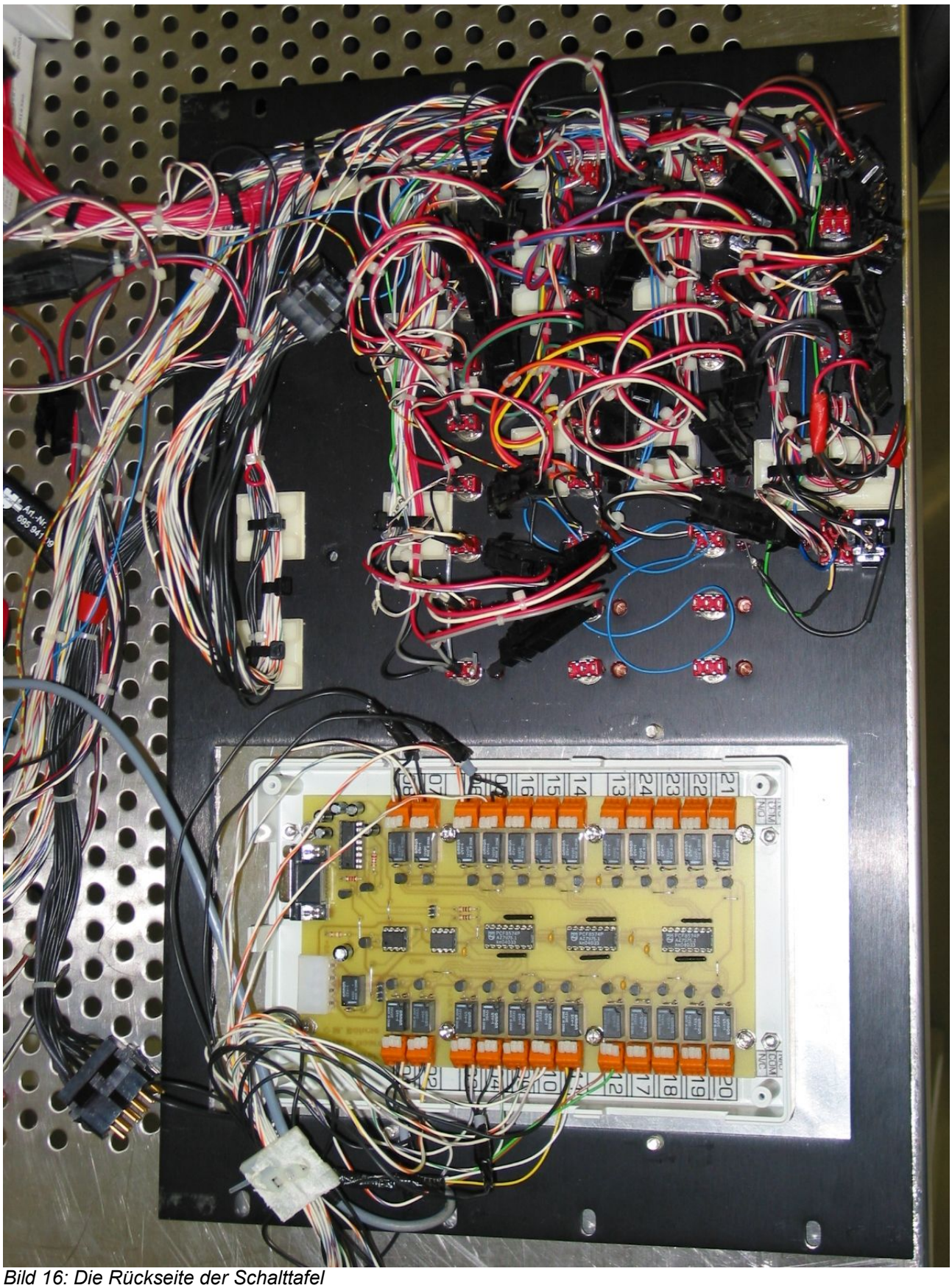

Bild 16: Die Rückseite der Schalttafel

Die folgende Tabelle enthält die Belegung des Steckverbinders zwischen Prozesssteuereinheit und Schalttafel:

Tabelle 5: Anschlussbelegung des Steckverbinders 1

Pinnummer	***Aderkennzeichnung***	***Bezeichnung und Relaiskontakt***
1	sw	GND
2	sw	GND
3	sw	GND
4	sw	GND
5	ws-gn-br	Ar vent, Stg.kanal 4, N/O
6	ws-bl	Quelle 1, Stg.kanal 8, N/O
7	sw	D1, Stg.kanal 11, COM
8	sw	über Freigabe gegen GND , unbenutzt
9	sw	D1, Schalter Panel, COM
10	sw	über Freigabe gegen GND , unbenutzt
11	ws-gr-br	H2O, Stg.kanal 7, N/O
12	ws-or-rt	O2, Stg.kanal 6, N/O
13	ws-gr	D2/D3 (Schalter), Stg.kanal 10, COM
14	ws-or-ge	NH3 vent, Stg.kanal 2, N/O
15	ws-or-gn	Quelle 2, Stg.kanal 9, N/O
16	ws-or-bl	unbenutzt
17	ws-or-li	NH3, Stg.kanal 1, N/O
18	ws-or-gr	H2, Stg.kanal 5, N/O
19	ws-li	Argon, Stg.kanal 3, N/O
20	ws-gr-sw	D2/D3 (Druckschalter), Stg.kanal 10, N/C

Die Verbindung der Prozesssteuereinheit zum Panel an der Anlage wird über ein fünf poliges Kabel hergestellt, mit folgender Belegung:

Tabelle 6: Anschlussbelegung des Steckverbinders 2

Pinnummer	***Aderkennzeichnung***	***Bezeichnung***
1	br	GND (geschaltet über Schalter 1/1)
2	Schirm	GND (geschaltet über Schalter 1/5)
3	gn	F1
4	ge	D1 Öffnen
5	ws	D1 Schließen

Technische Daten

Abwärtszähler

- Versorgungsspannung : 15 V bis 36 V Gleichspannung
- maximale Stromaufnahme : 500 mA
- Optokoppler
  - Kollektor-Emitter-Spannung : V_{CEO} = 30 V
  - Kollektorstrom : I_C = 100 mA
  - Verlustleistung : P_V = 200 mW

Prozesssteuereinheit

- Versorgungsspannungen : 5 V ± 10% ; 12 V ± 10%
- maximale Stromaufnahme : 5 V : 110 mA ; 12 V : 300 mA
- Ausgangsrelais
  - maximale Schaltspannung : 125 VAC / 60 VDC
  - maximale Schaltleistung : 62,5 VA / 30 W
  - maximaler Schaltstrom : 0,5 A @ 125 VAC ; 1 A @ 24 VDC

7 Abbildungs- und Tabellenverzeichnis

8 Literaturquellen

[1] Markku Leskelä, Mikko Ritala: Chemie der Atomlagenabscheidung (Atomic Layer Deposition): jüngste Entwicklungen. *Angew. Chemie* **115** (48), Seiten 5706 bis 5713 (2003)

[2] Firmenschrift: Instruction manual - Tymgard Process Controller. Tystar Corporation, 1984

[3] Firmenschrift: CMOS 4522. Philips Semiconductors, 1995

[4] Firmenschrift: CMOS 4511. Motorola, 1995

[5] Firmenschrift: The I²C-Bus Specification. Philips Semiconductors, 2000

[6] Himpe,Vincent: The I2C FAQ V1.3. Internet (http://neptune.lisa.univ-paris12.fr/Electronik/F_I2C.htm), 10.12.2004

[7] Firmenschrift: PCF8574. Philips Semiconductors, 1997

[8] Firmenschrift: MAX232. Maxim, 1997

[9] Firmenschrift: MAX813. Maxim, 1995

[10] Firmenschrift: NE555. Texas Instruments, 2004