Projektpraktikum

Mikrocontroller - Steuerung einer Alarmanlage

Deichstetter Harald
9555258

Projektbetreuer : **Prof. Dr. Hanspeter Mössenböck**
Ing. Anton Kral

Inhaltsangabe:

1. Aufgabenstellung

Es ist eine Alarmanlage zu entwickeln, die von einem Mikrocontroller zu steuern bzw. zu überwachen ist. Die Anlage soll mehrere Räume überwachen und über Türkontakte betrieben werden.

Zumindest folgende Einstellungen sollen möglich sein:

- einzelne Räume aktivieren/deaktivieren,
- Ein/Ausschalten der Anlage zu einer vorgegebenen Zeit.
- Verwaltung mehrerer Benutzer, welche mittels Codes Zutritt haben.

Die Codeeingabe bzw. Änderung erfolgt mittels eines am Mikrocontroller angeschlossenen Tastenblocks + Anzeige.

Plattform: Windows, Mikrocontroller 80C161, C

2) Einführung

Das schnell wachsende Gebiet der zeitkritischen Steuerungsapplikationen stellt eines des wichtigsten Anwendungen für Mikrocontroller dar. Komplexe Kontrollalgorithmen arbeiten mit einer Vielzahl digitaler und auch analoger Eingabesignale, wobei die entsprechenden Antwortzeiten darauf nur innerhalb bestimmter Grenzen liegen dürfen.
Oft sind auch Platzbedarf, Stromverbrauch und Systemkosten von entscheidender Bedeutung.

Es werden daher Mikrocontroller benötigt welche...
- ein hohes Maß an Systemintegration,
- geringe Notwendigkeit für zusätzliche externe Einheiten und dem damit verbundenen Software/Hardware - Aufwand,
- und hohe Systemsicherheit besitzen

Mit zunehmender Komplexität der Anwendungen entstanden hohe Anforderungen an die CPU und der integrierten Baugruppen welche sich nicht mehr mit konventionellen 8-Bit-Controllern erreichen ließen. Um diese Leistungen zu erbringen, wurden 16-Bit-CMOS Mikrocontroller entwickelt. Dabei wurde besonders darauf geachtet, daß eine leichte und schnelle Umstellung auf Controller späterer Generationen mit unterschiedlichen Speichergrößen, erweiterten Baugruppen und verschiedener Anzahl von Ein /Ausgängen möglich ist.
Da Programme für Steuerungsanwendungen immer umfangreicher werden, kommen bei der Programmierung höhere Sprachen wie C, Modula, Forth zum Einsatz.

Mit dem 80C166 entstand die ersten Generationen der 16-Bit-Controller-Familie. Diese Geräte haben die C166-Architektur gegründet.
Die C165- und C167-Typen sind Mitglieder der zweiten Generation. Sie zeichnet sich durch mehr Anweisungen, erhöhten Adreßbereich und zusätzlichen internen Speicher aus.
Wobei der C165 eine reduzierte Version des C167 ist, und einen niedrigeren Stromverbrauch auf Kosten des A/D-Umsetzers, der CAPCOM-Einheiten und des PWM - Modul aufweist.
Die noch einmal reduzierten C161-Derivate eignen sich besonders für kostenempfindliche Anwendungen.

Überblick grundlegender Merkmale des C161

Der C161-Controller ist ein kostensparender Vertreter der 16-Bit Siemens Mikrocontroller-Familie. Er verbindet hohe CPU-Leistung (bis zu 8 Millionen Anweisungen / Sekunde) mit hoher Leistung der internen Baugruppen.

Einige Merkmale des C161:

16-Bit-CPU mit 4 stufiger-Pipeline
- 125 ns minimale Ausführungszeit, die meisten Anweisungen werden in 1 Zyklus abgearbeitet
- 625 ns Multiplikation (16 Bit * 16 Bit), 1.25 ys Division (32-Bit/16-Bit)
- mehrfache Datenbusse
- 4 MByte linearer Adreßbereich für Code und Daten (von Neumann Architektur)
- Systemstack mit automatischer Über/Unterlauferkennung

Effizienter Befehlsvorrat
- Bit, Byte und Wort Datentypen
- flexible Adressierungsarten
- erweiterte boolsche Bitmanipulation
- Hardware-Traps um Fehler während der Laufzeit zu identifizieren.

Integrierter Speicher
- interner RAM für Variablen, Registerbänke, Systemstack und Code (2 kByte am C161O, 1 kByte auf C161V, C161K)
- interner Maskenprogrammierbarer ROM, oder Flash-Speicher

Bus-Interface
- gemultiplexte oder gedemultiplexte Buskonfiguration
- Chip-Select-Generierung
- 8 oder 16 Bit Datenbusse
- programmierbare Bus-Characteristik für fünf Adreßbereiche

16-stufiges Interrupt System
- 20/14 Interrupts (C161O/C161V/K) mit getrennten Interruptvektoren
- 315/625 ns maximale Verzögerung bis zur Abarbeitung des Interrupts
- externe Interrupts

Interne Baugruppen
- multifunktionelle allgemeine Zeitgeber - Einheiten
  GPT1: drei 16-Bit-Zeitgeber/Zähler, 500-ns minimale Taktzeit
  GPT2: zwei 16-Stück-Zeitgeber/ Zähler, 250-ns minimale Taktzeit (nur C161O)
- asynchrone/synchrone serielle Schnittstelle mit Baud-Raten-Generator, Parität, und Überlauferkennung.
- Wachdog-Timer mit programmierbaren Zeitintervallen
- Bootstrap Loader für flexibele Systeminitialisierung (nicht C161V)

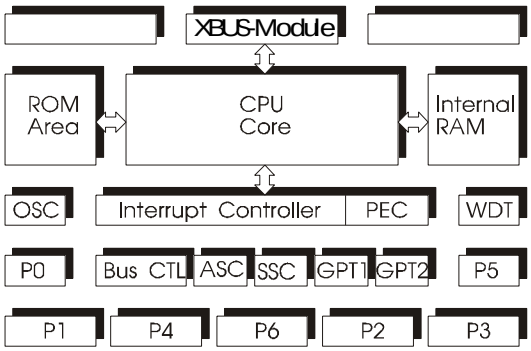
63 IO-Pins mit individueller Bit-Adressierungsmöglichkeit
- Tri-state (gleichzeitige Ein/Ausgabe möglich)

Siemens-CMOS-Prozeß
- geringer Stromverbrauch, Strom-Spar-Einrichtung

Die nachfolgende Tabelle beschreibt die Unterschiede zwischen den drei Typen, welche hier beschrieben wurden

Feature	C161V	C161K	C161O
interner Speicher	1 Kbyte	1 KByte	2 Kbytes
Chip Select Signale	---	2	4
Bus Modes	MUX	MUX / Demux	MUX / Demux
Strom-Spar-Modus	---	ja	ja
Fast External Interrupts	4	4	7
GPT1	ja	ja	ja
Ein / Ausgabefunktionalität des GPT1	---	ja	ja
GPT2	---	---	ja
Bootstrap Loader	---	ja	ja

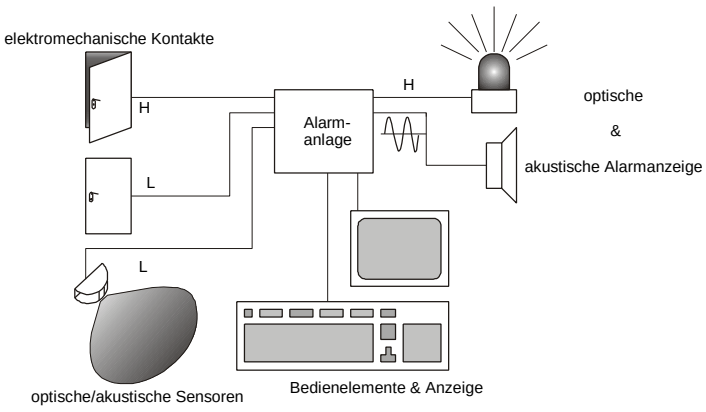
Block-Diagramm eines C161:



ASC . . . . . Asynchronous/synchronous Serial Controller
BUS CTL . . . . Bus Controller
CPU . . . . . Central Processing Unit
GPT . . . . . General Purpose Timer unit
PEC . . . . . Peripheral Event Controller
RAM . . . . . Random Access Memory
ROM . . . . . Read Only Memory
SSC . . . . . Synchronous Serial Controller
WDT . . . . . Watchdog Timer
XBUS . . . . . Internal representation of the External Bus

Anwendung eines C161-Mikrocontrollers zur Steuerung einer Alarmanlage

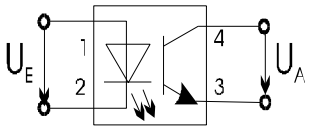
Alarmanlagen werden verwendet, um auf Störungen, Gefahren oder unbefugtem Eindringen automatisch zu reagieren und durch Warngeräte wie Hupen, Sirenen, Blinkleuchten auf sich aufmerksam zu machen. Anwendungen sind zum Beispiel Feuermeldeanlagen, Polizeirufanlagen oder Raumschutzanlagen(Überfall- und Einbruchmeldeanlagen). Bei Raumschutzanlagen werden hauptsächlich Detektoren verwendet, die elektromechanisch (Kontakte), akustisch (z.B. Ultraschall), optisch (z.B. Infrarotschranken) oder durch Feldänderung (elektr. Felder zwischen Elektroden) arbeiten. Neuere Systeme werden auch mittels an Rechner angeschlossenen Video-Kameras realisiert.
Um eine einfache Alarmanlage zu realisieren, würde es genügen einen Sensor mit einem Warngerät zu verbinden. Diese würde bei jedem Ansprechen des Sensors aktiviert werden und einen Alarm auslösen.
Häufig werden aber bestimmte Anforderungen an das Gerät gestellt. So soll diese z.B bestimmten Personen den Zutritt gewähren, oder sich zu einstellbaren Tageszeiten ein und ausschalten. Einzelne Sensoren sollen über eine einfach zu bedienende Oberfläche vollständig deaktiviert, oder Eintrittscodes vergeben werden. Für solche Anwendungen sind Mikrocontroller sehr gut geeignet. Durch den Einsatz von externen Interrupts kann auf Änderungen der Sensoren schnell reagiert und über Portleitungen Warngeräte, falls nötig, angesprochen werden. An den vorhandenen Datenbus können Komponenten wie eine Anzeige, die Tastatur oder Zeitgeber einfach angeschlossen werden.
Das nachfolgende Blockschaltbild zeigt den grundsätzlichen Aufbau einer Alarmanlage:



3) Hardwareaufbau

Die Kopplung der einzelnen Hardwarekomponenten sind im Blockschaltbild auf Seite 6 ersichtlich.
Um einen sicheren Betrieb zu gewährleisten, müssen die vom Benutzer zugänglichen Komponenten speziell gesichert werden. Dies geschieht mittels Galvanischer Trennung. Üblicherweise wird diese mittels Optokoppler realisiert.

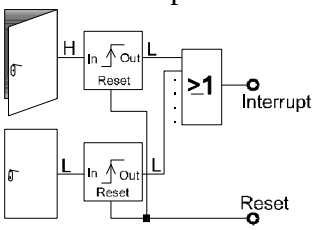
Ein Optokoppler funktioniert folgendermaßen:



Eingehende Signale erzeugen an der Leuchtdiode einen Lichtimpuls. Dieser wird von einem lichtempfindlichen Transistor erkannt und schaltet durch. D.h er wird leitfähig. Dadurch können Störungen am Eingang im schlechtesten Fall die Leuchtdiode und nicht den Transistor und weitere daran angeschlossene Komponenten zerstören.
Ein weiterer Vorteil von Optokopplern ist, daß am Ein- und am Ausgang getrennte Versorgungsspannungen verwendet werden können. (Bei Verwendung eines einfachen Transistors würde man ein gemeinsames Bezugspotential benötigen). Bricht nun die Versorgung an den externen Komponenten zusammen, z.B. durch einen Kurzschluß, so wirkt sich dies nicht auf den Mikrokontroller aus.

Im diesem Projekt wurden damit alle Leitungen zu den Türen, wie auch die zur Abfrage der Tastatur gesichert. Gehen Signale aus der Alarmanlage über die Optokoppler hinaus, so müssen diese zusätzlich verstärkt werden. (Die Leuchtdiode benötigt für den Betrieb 10 mA). Dies geschieht über Bustreiber.
Die Anzeige mußte speziell behandelt werden, da die Optokoppler offenbar nicht in der Lage waren, diese richtig anzusprechen. Anstelle von Optokopplern wurden deshalb Transistoren verwendet. Auch diese werden zusätzlich über Bustreiber angesprochen.
Eine genaue Beschaltung der Optokoppler/Bustreiber ist auf den Schaltplänen ersichtlich.

Um eine genaue Uhrzeit zu erhalten, wurde anstatt eines internen Timers des 80C161 zusätzlich ein externer Uhrenbaustein (IC28, RTC72421A) verwendet. Dieser ist über den Adress- und Datenbus ansprechbar.



Ein Alarm wird durch Ansprechen eines externen Interrupt ausgelöst. Dazu müssen alle Türen ODER-verknüpft werden. Bei einfacher ODER-Verknüpfung würde das Problem auftreten, daß falls eine Türe ständig geöffnet ist, das Öffnen einer weiteren nicht mehr erkannt werden kann. D.h Der Alarm ist immer aktiv. (H∨Tür1∨Tür2...=H)
Als Abhilfe wurden positiv flankengetriggerte Flip-Flops (FF) dem ODER-Gatter vorgeschaltet.

➔ Wird eine Tür geöffnet so wird das FF gesetzt. Der yC erkennt dies über die Verknüpfung und löst je nach Türstatus (ob sie aktiviert oder deaktiviert ist) einen Alarm aus. Danach wird das FF wieder gelöscht. Damit kann auch bei einer ständig geöffneten Tür noch ein Impuls für den externen Interrupt erzeugt werden.

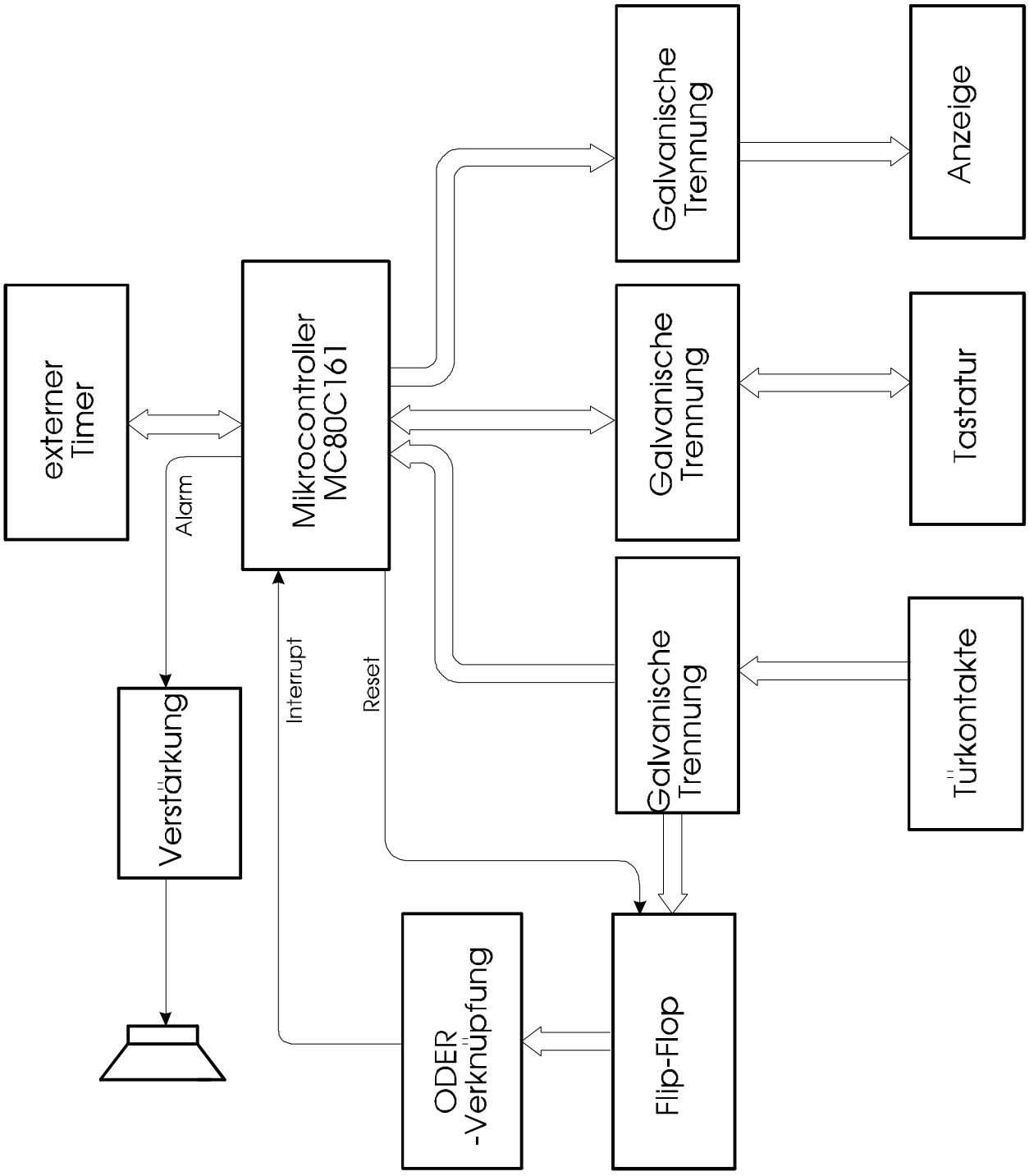
Das Signal für den Lautsprecher wird derzeit nur mittels eines einfachen Transistors verstärkt. Es existiert aber zusätzlich noch ein Anschluß, so daß ein externer Verstärker angeschlossen werden kann.

Beim 80C161 sind weniger als 3 Ports ansprechbar. Alle anderen sind durch alternative Funktionen bereits belegt (serielle Schnittstelle...). Benötigt werden aber 16 Leitungen zur Abfrage der Türen, ext. Interruptleitungen, 8 Tastaturabfrageleitungen ...). Port 1 wird zusätzlich standardmäßig für die Adressierung des externen Speichers benutzt.
Gelöst wurde das Problem durch Multiplexen der Adressleitungen. D.h Port 1 kann zusätzlich für Steuerleitungen verwendet werden. Möglich ist dies durch Umsetzen der Jumper am Mikrocontrollerboard (siehe Seite 16, 80C161 - Jumperbelegungen). Zusätzlich muß diese Änderung auch im Bustiming berücksichtigt werden.

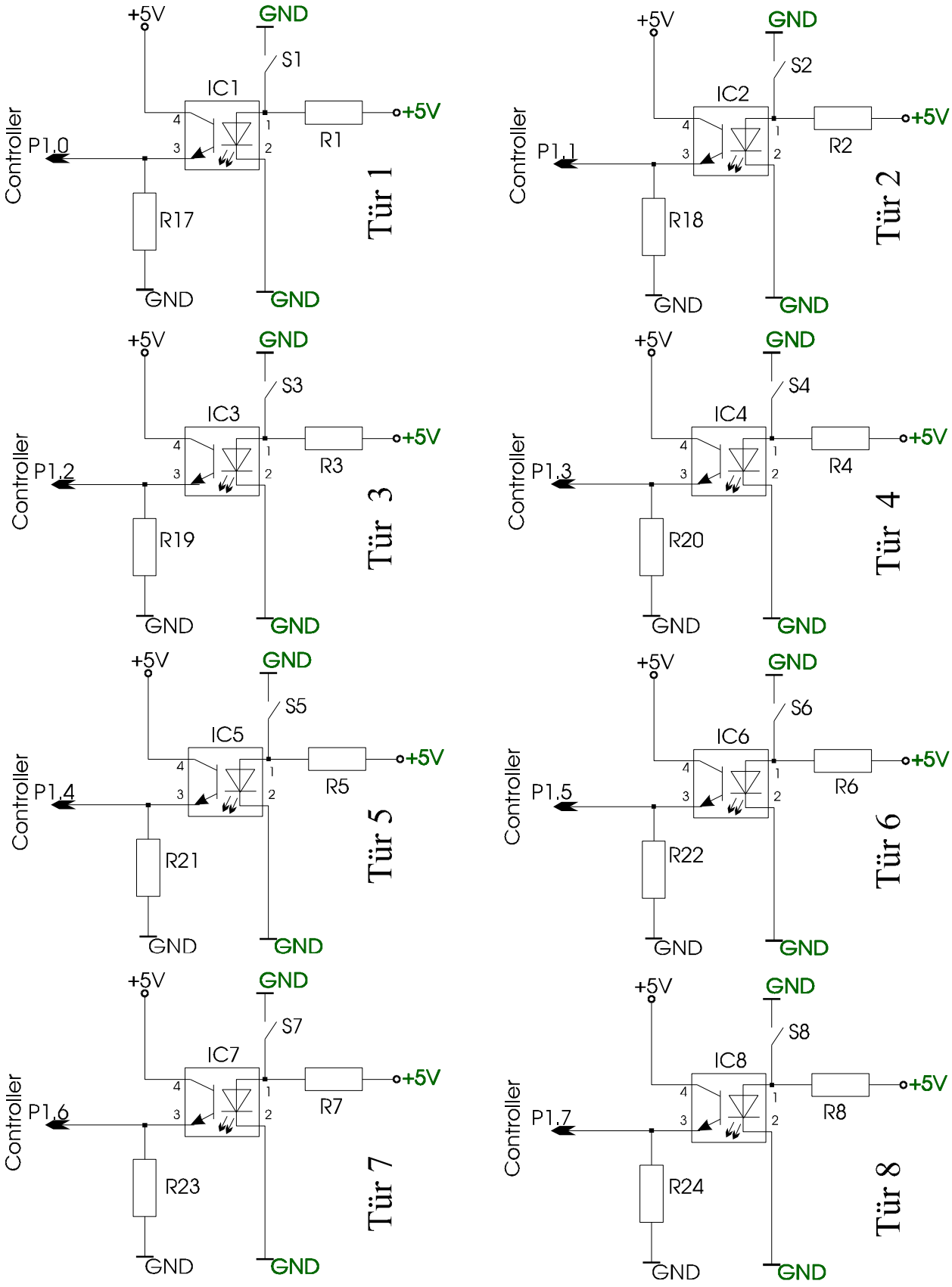
Die Tastatur besitzt 4 Ein/4Ausgangsleitungen. Zur Abfrage der gedrückten Taste wird nacheinander an den Zeilenleitungen ein Signal angelegt (100 HZ), und anschließend die Spalten abgefragt. Je nach Zeilen und erhaltenen Code an der Spalte kann daraus die Taste ermittelt werden.

Da sich Anzeige und Tastatur weit entfernt vom Mikrocontroller befinden, müssen die Leitungen speziell gegen hochfrequente Störungen (Rückkopplungen , Einstreuungen) gesichert werden. Deshalb wird die Anzeige nicht mehr, wie ursprünglich vorgesehen, über den Adress/Datenbus angesprochen, sondern es werden nun einige Portanschlüsse dafür verwendet. Dadurch ist es möglich die Taktrate der zu übertragenden Daten gering zu halten. Zusätzlich wird am Eingang der Anzeige ein RC-Tiefpaß + Bustreiber mit Schmitttriggereingang zum Herausfiltern der hohen Störfrequenzen verwendet. (siehe Berechnung)

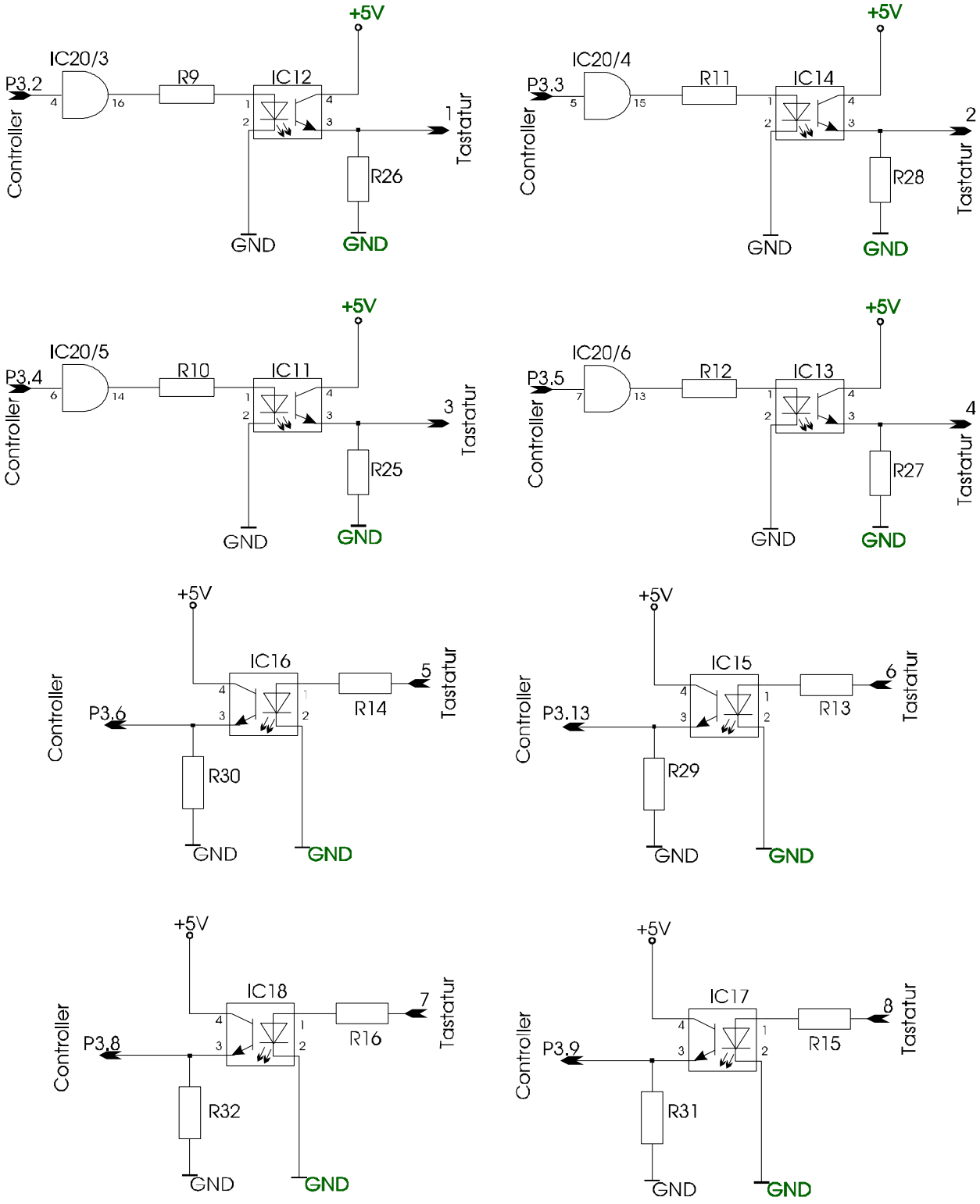
Blockschaltbild



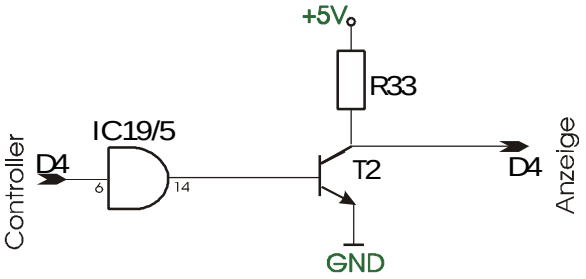
Türkontakte

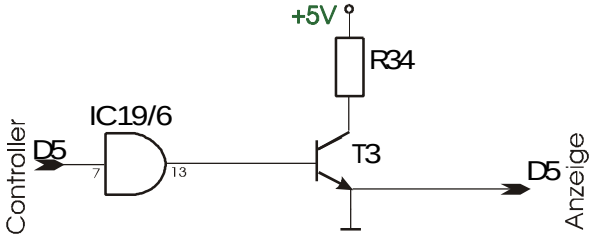


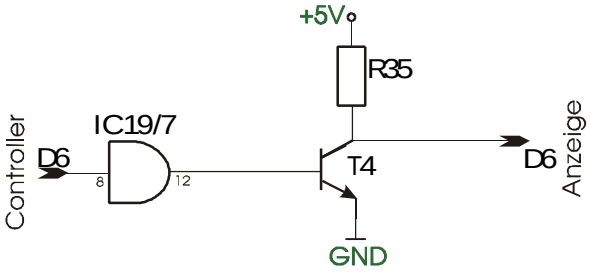
Tastatur

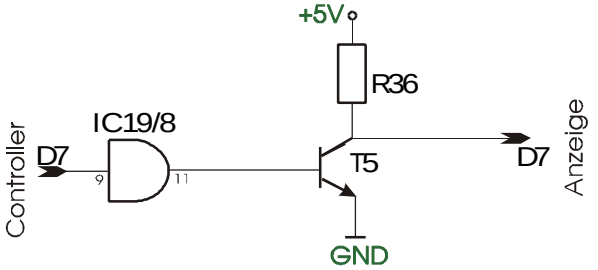

+5V
IC20/3
P3.2
R9
IC12
Controller
Tastatur
1
R26
GND
GND
+5V
IC20/4
P3.3
R11
IC14
Controller
2
R28
Tastatur
GND
GND
+5V
IC20/5
P3.4
R10
IC11
Controller
3
R25
Tastatur
GND
GND
+5V
IC20/6
P3.5
R12
IC13
Controller
4
R27
Tastatur
GND
GND
+5V
IC16
Controller
P3.6
R14
5
Tastatur
R30
GND
GND
+5V
IC15
Controller
P3.13
R13
6
Tastatur
R29
GND
GND
+5V
IC18
Controller
P3.8
R16
7
Tastatur
R32
GND
GND
+5V
IC17
Controller
P3.9
R15
8
Tastatur
R31
GND
GND

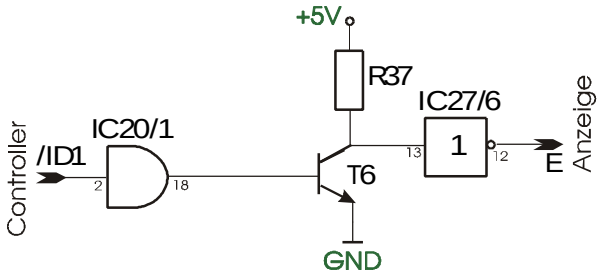
Anzeige

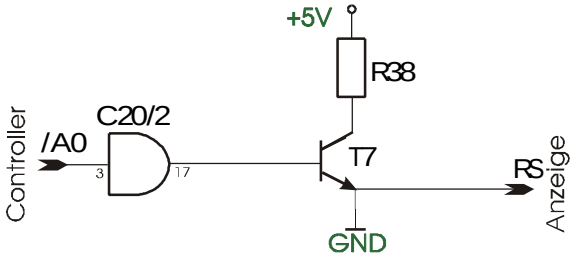

+5V
R33
IC19/5
Controller
D4
6
14
T2
D4
Anzeige
GND


+5V
R34
IC19/6
Controller
D5
7
13
T3
D5
Anzeige

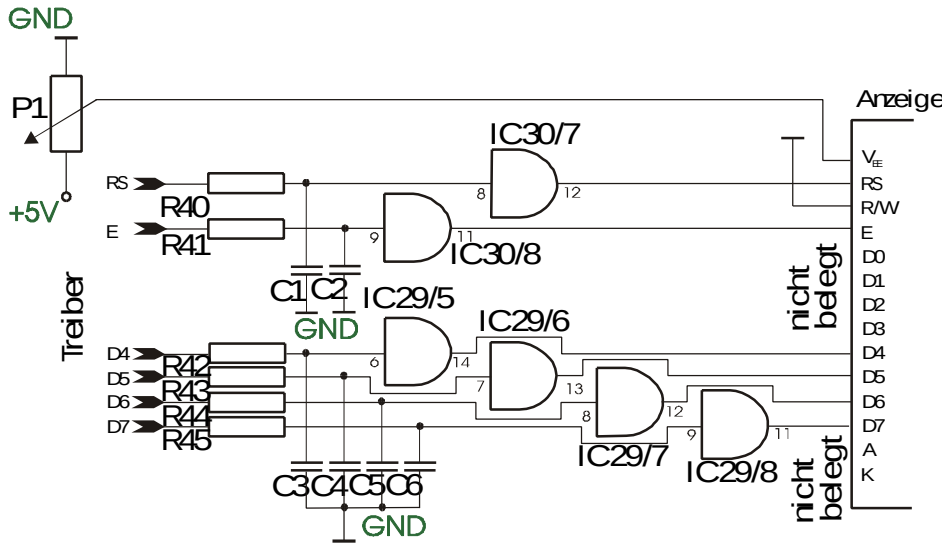

+5V
R35
IC19/7
Controller
D6
8
12
T4
D6
Anzeige
GND


+5V
R36
IC19/8
Controller
D7
9
11
T5
D7
Anzeige
GND

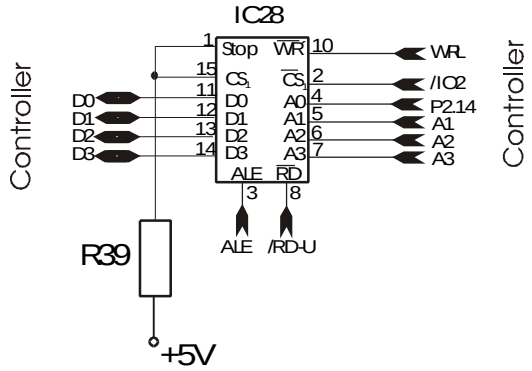

+5V
R37
IC27/6
IC20/1
Controller
/ID1
2
18
13
1
12
T6
E
Anzeige
GND


+5V
R38
C20/2
Controller
/A0
3
17
T7
RS
Anzeige
GND

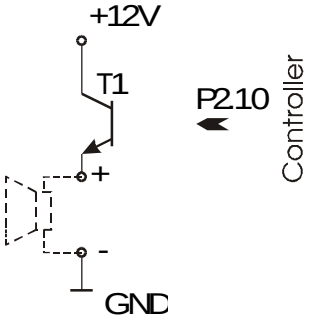
Anzeige - Fortsetzung



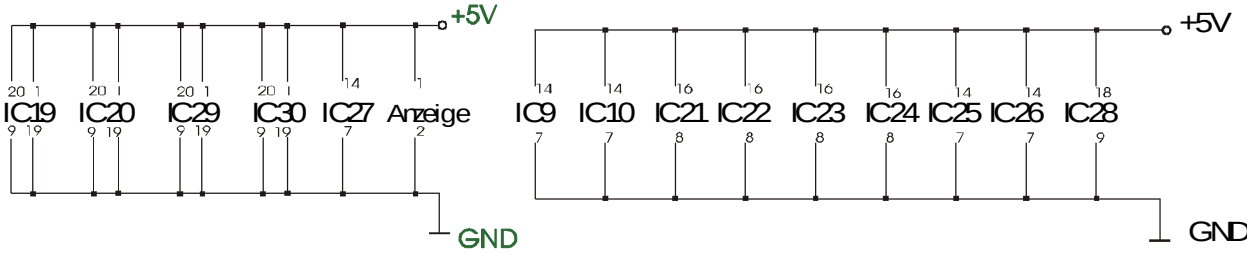
Uhrenbausteinbeschaltung



Lautsprecheransteuerung

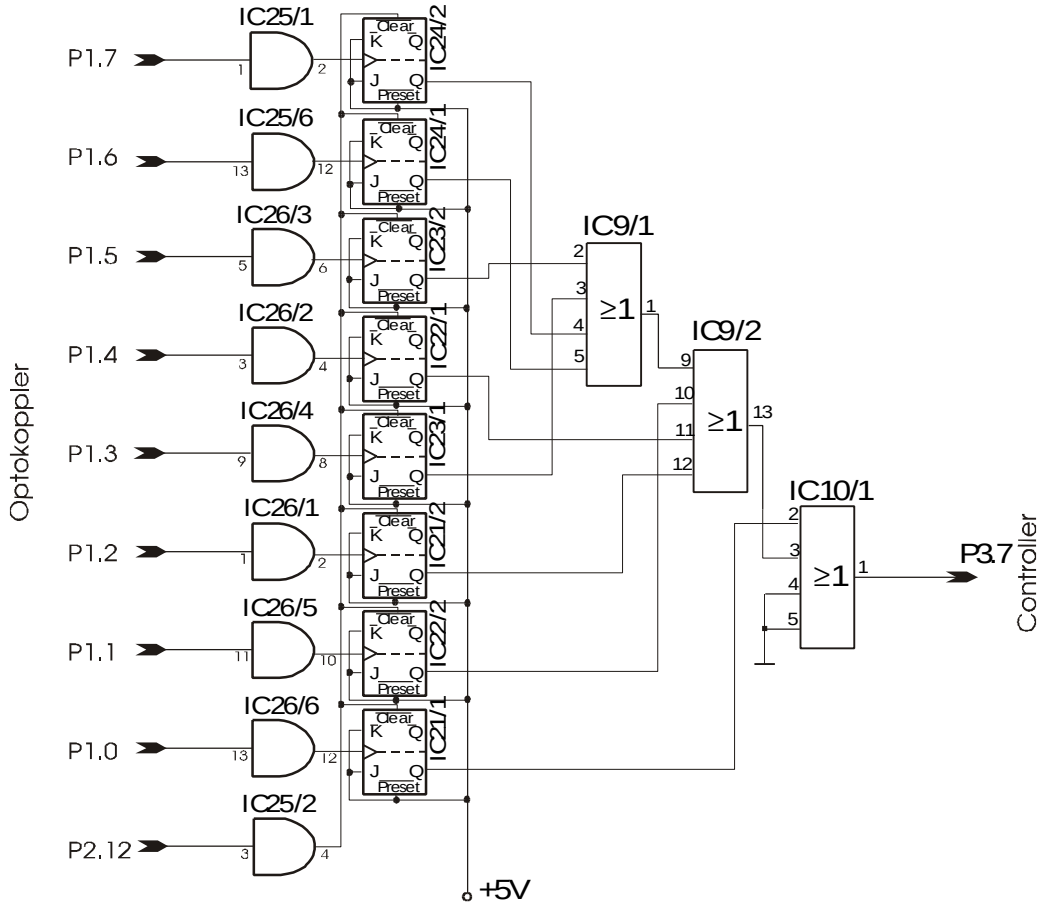


Versorgungsplan



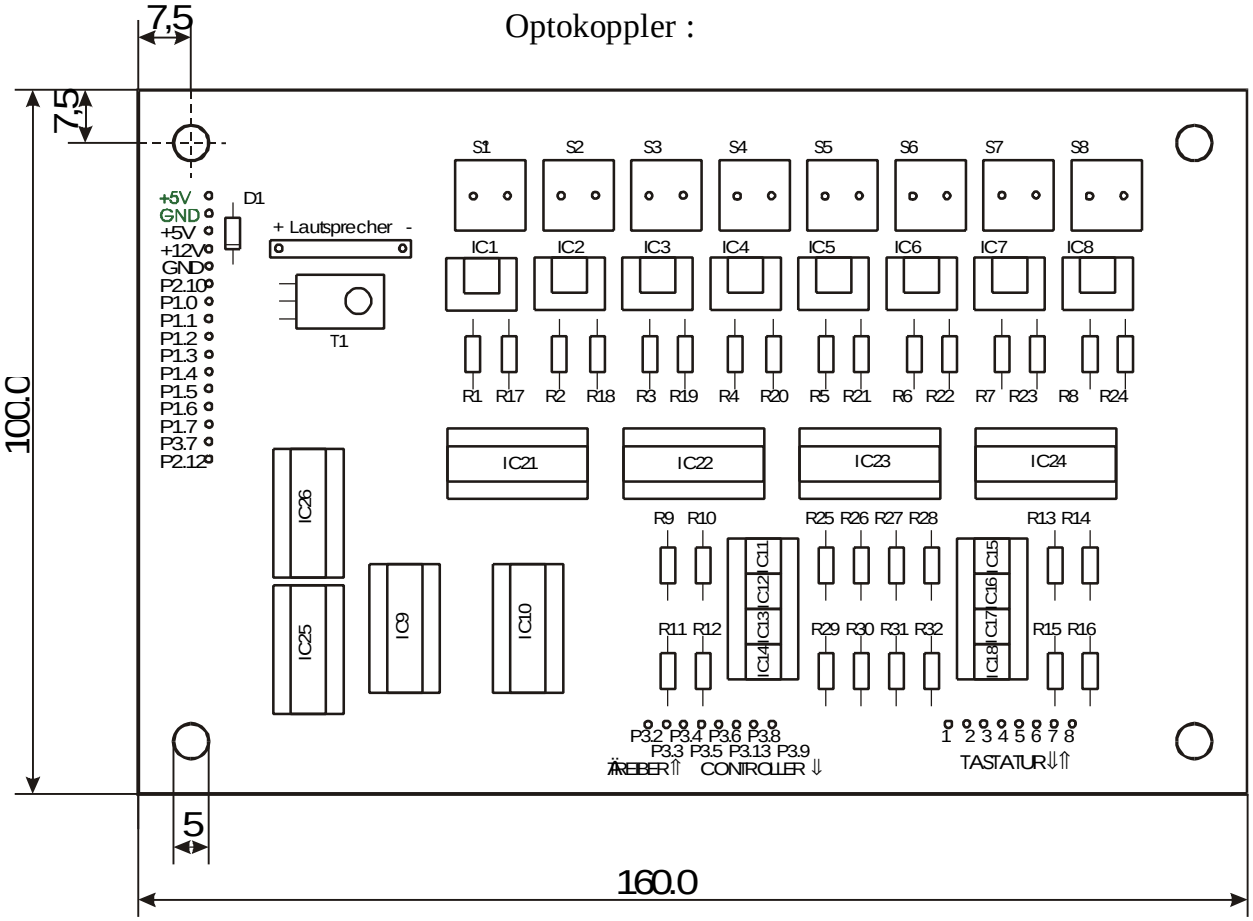
... Versorgung außerhalb ... Versorgung fuer Mikrocontroller

Interruptansteuerung

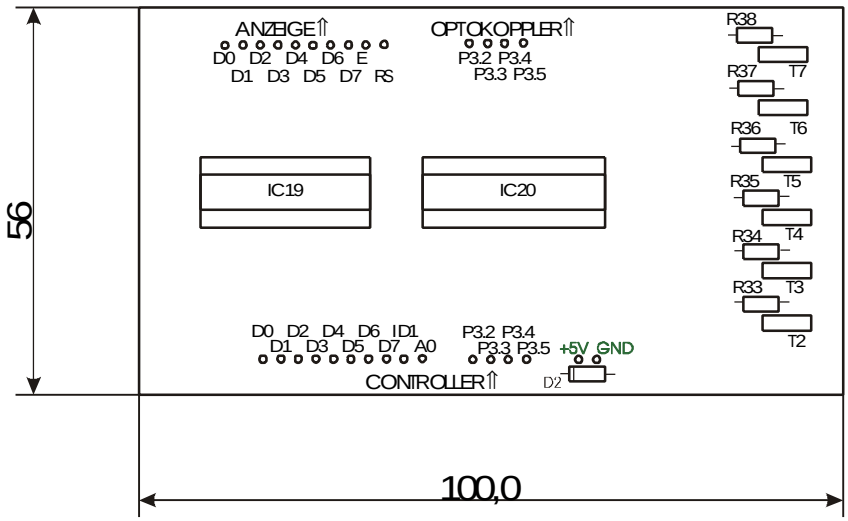


Bestückungsplan

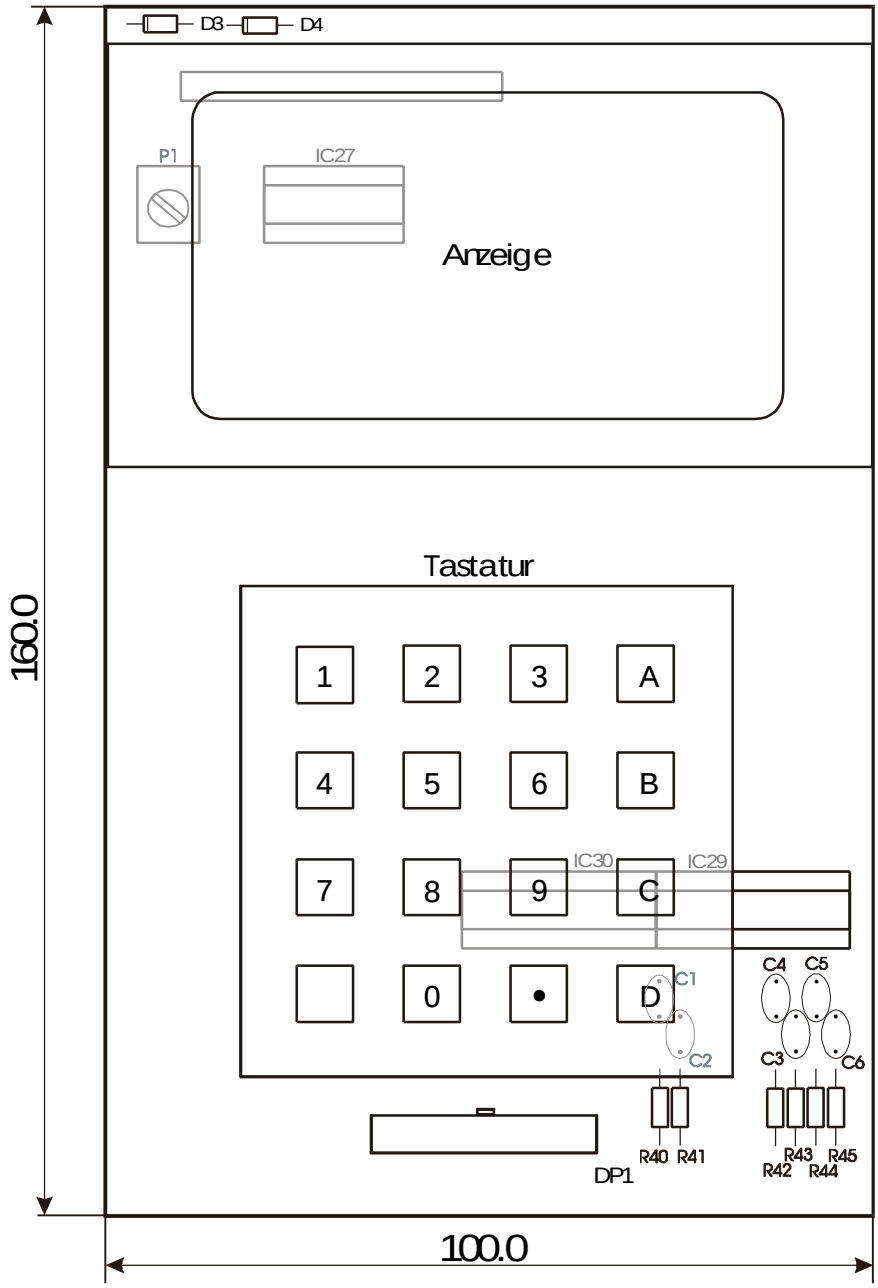
Optokoppler :



Treiber :

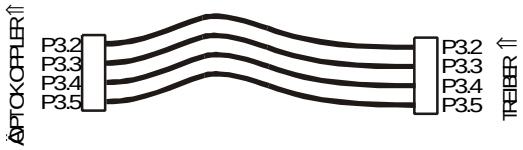


Anzeige & Tastatur

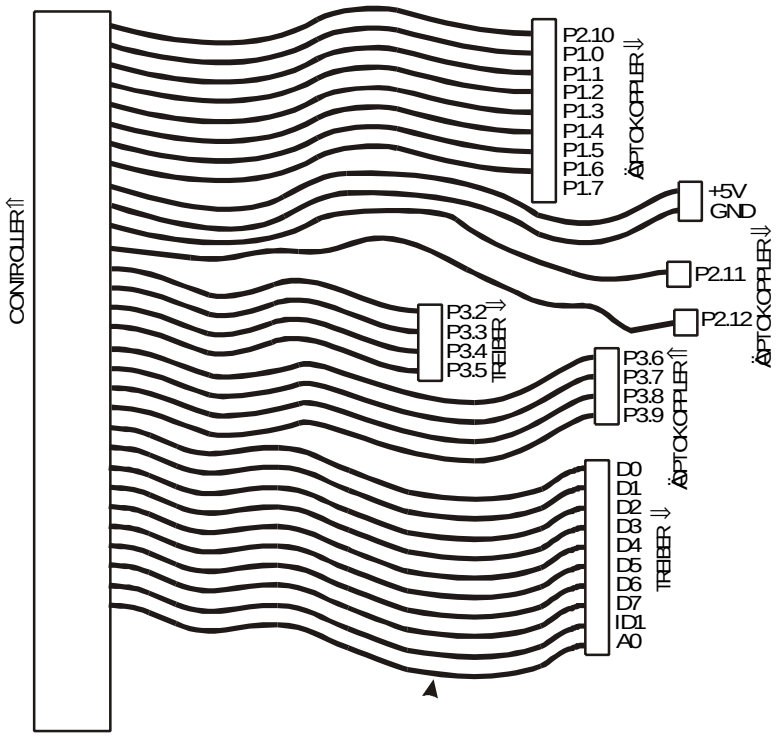


Anschlüsse

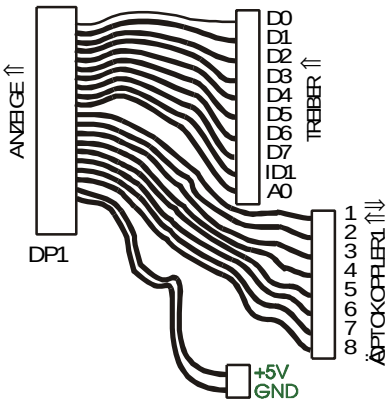
Tastatur :



Controller :

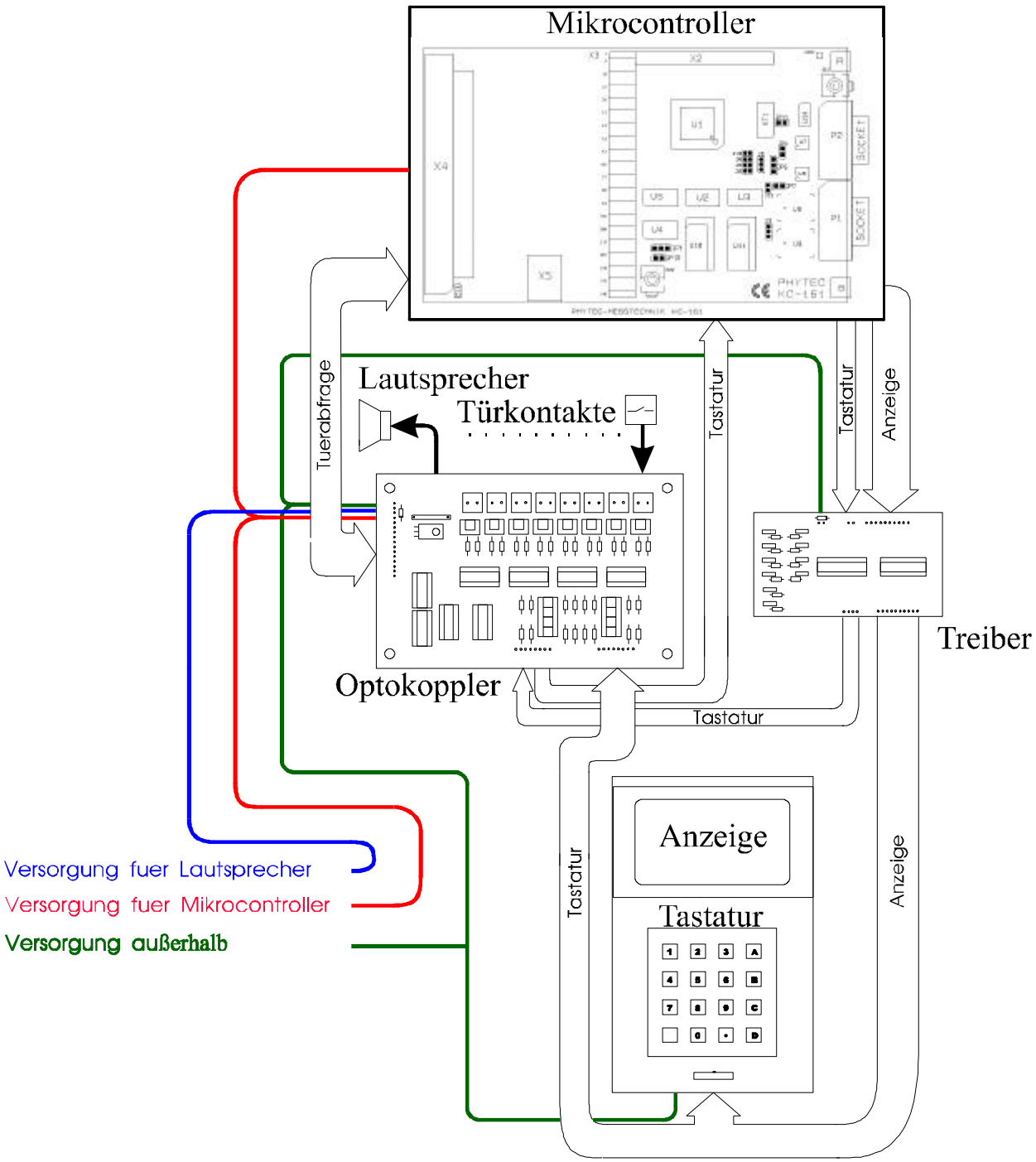


Anzeige & Tastatur :

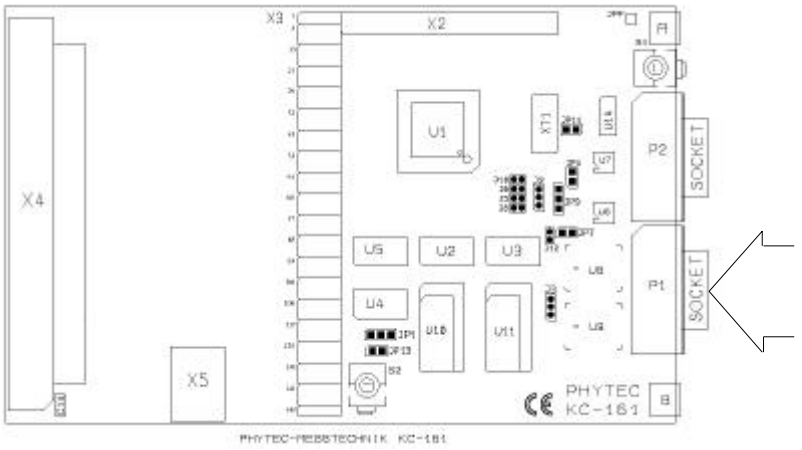


jeweils für 4 und 8 Bit

Anschlüsse - Fortsetzung



Einstellungen des C161 - Controllers



	Default Setting	Alternate Setting
JP 1	(2+3) demultiplexed operation	(1+2) multiplexed operation
J 2	(1+2) external ROM/Flash active	(2+3) internal ROM/Flash active
J 3	(2+3) 256 or 64 KB RAM	(1+2) 1 MB RAM
JP 4	(closed) Mode 0	(open) Mode 1
J 5	(open) Emulation Mode disabled	(closed) Emulation Mode enabled
J 6	(open) Adapt-Mode enabled	(closed) Adapt-Mode disabled
JP 7	(closed) no Chip-Selects active	(open) all Chip-Selects active
J 8	(closed) Segment-address A16-A23 active	(open) Segment-address A16, A17 active
JP 9	(1+2) Bootstrap Mode Execution Mode	(2+3) normal Program
J 10	(closed) System Clock Speed = XTAL1:1	(open) System Clock Speed = XTAL1:2
JP 11	(open) P2.9 freely available	(closed) P2.9 as RS-232 Debug-Interface
J 12	(closed) WRL+WRH disabled	(open) WRL+WRH enabled
JP 13	(open) internal Bootstrap-Loader disabled	(closed) internal Bootstrap-Loader enabled

.. für Alarmanlage benutzt

KitCON-161 - Anschlußbelegung

	Pin							
Versorgung	1	VCC	2	VCC	3	GND	4	GND
Datenbus	5	D0	6	D2	7	D4	8	D6
	9	D1	10	D3	11	D5	12	D7
	13	D8	14	D10	15	D12	16	D14
	17	D9	18	D11	19	D13	20	D15
Address-Bus	21	A0	22	A2	23	A4	24	A6
	25	A1	26	A3	27	A5	28	A7
	29	A8	30	A10	31	A12	32	A14
	33	A9	34	A11	35	A13	36	A15
	37	A16	38	A18	39	A20	40	
	41	A17	42	A19	43	A21	44	
Controll-Signale	45	/RD-U	46	/RD-U	47	/RESO-U	48	/RES-U
	49	/WRL	50	ALE	51	/EA	52	/NMI/U
	53	/CS0(P60)	54	/CS2(P62)	55	/IO1	56	
	57	/CS1(P61)	58	/CS3(P63)	59	/IO2	60	
	61		62		63		64	
	65		66		67		68	
	69		70		71		72	
	73		74		75		76	
	77		78		79		80	
	81		82		83		84	
	85		86		87		88	
	89		90		91		92	
Digitaler Port 2	93		94	P2.10	95	P2.12	96	P2.14
	97	P2.9	98	P2.11	99	P2.13	100	P2.15
Digitaler Port 3	101		102	P3.2	103	P3.4	104	P3.6
	105		106	P3.3	107	P3.5	108	P3.7
	109	P3.8	110	P3.10	111	/WRH	112	
	113	P3.9	114	P3.11	115	P3.13	116	
Digitaler Port 1	117	P1.0	118	P1.2	119	P1.4	120	P1.6
	121	P1.1	122	P1.3	123	P1.5	124	P1.7
	125	P1.8	126	P1.10	127	P1.12	128	P1.14
	129	P1.9	130	P1.11	131	P1.13	132	P1.15
	133		134		135		136	
	137		138		139		140	
	141		142		143		144	
	145		146		147		148	
Versorgung	149	VCC	150	VCC	151	GND	152	GND

...fur Erweiterung reserviert ...belegt

P1.0	Tür 1
P1.1	Tür 2
P1.2	Tür 3
P1.3	Tür 4
P1.4	Tür 5
P1.5	Tür 6
P1.6	Tür 7
P1.7	Tür 8
P1.8	Tür 9
P1.9	Tür 10
P1.10	Tür 11
P1.11	Anzeige (RS)
P1.12	Anzeige (D4)
P1.13	Anzeige (D5)
P1.14	Anzeige (D6)
P1.15	Anzeige (D7)

P2.9	Anzeige (E)
P2.10	Lautsprecher
P2.12	FF-Ruecksetzen
P2.14	Timer (A0)
P3.2	Tastatur (Pin 1)
P3.3	Tastatur (Pin 2)
P3.4	Tastatur (Pin 3)
P3.5	Tastatur (Pin 4)
P3.6	Tastatur (Pin 5)
P3.7	Tür - Interrupt
P3.8	Tastatur (Pin 7)
P3.9	Tastatur (Pin 8)
P3.13	Tastatur (Pin 6)

D0	Timer (D0)
D1	Timer (D1)
D2	Timer (D2)
D3	Timer (D3)
/IO2	Timer (/CS)
ALE	Timer (ALE)
/RD-U	Timer (/RD)
/WRL	Timer (/WR)
A1	Timer (A1)
A2	Timer (A2)
A3	Timer (A3)

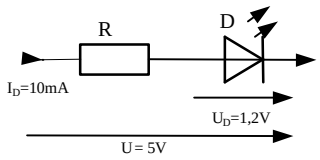
Berechnung der verwendeten Bauteile:

Pull-Down/Up Widerstände werden seriell am Ausgang der Transistoren verwendet, um ihn beim Durchschalten nicht zu überlasten. (Ausgang des Transistors würde direkt an der Versorgung liegen) Die Widerstandsgrößen sind allgemeingültige Werte und wurden einem Datenbuch entnommen.
Pull-Down Widerstände: 10 kΩ, Pull-Up Widerstände: 470 Ω

R_{17}-R_{32} = 10kΩ

R_{33}-R_{38} = 470Ω

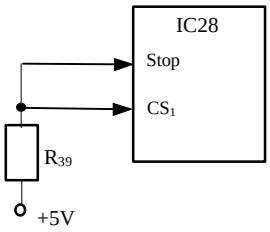
Serienwiderstand zur Leuchtdiode (im Optokoppler)
Die verwendete Leuchtdiode im Optokoppler benötigt eine Durchlaßspannung von 1,2V bei einem Strom von 10mA. Diese Werte wurden einem Datenbuch entnommen. Um die Leuchtdiode mit 5V betreiben zu können, muß die restlich Spannung an einem Widerstand verbraucht werden.



$R = {}^{U}/_{I} = {}^{(5V-1,2V)}/_{10mA} = 380\Omega$
⇨ gewählt : 270Ω ±10% (E12)

R_1-R_{16} = 270Ω

Pull-Up Widerstand am Timer-Baustein :

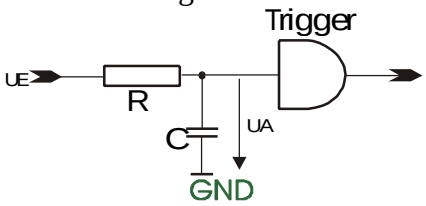


R_{39}= 4,7 kΩ
(laut 80C535 Platine)

R_{39} = 4,7kΩ

Berechnung des Tiefpaß-Filters:

Mit einfachen Kombinationen aus Widerständen und Kondensatoren lassen sich Übertragungsglieder aufbauen, die man als frequenzabhängige Spannungsteiler auffassen kann. Ein RC-Glied mit dem unten gezeigten Aufbau bildet einen Tiefpaß. Bis zu einer einstellbaren Frequenz werden Schwingungen vom Eingang zum Ausgang gut übertragen, oberhalb dieser jedoch mit steigender Frequenz zunehmend gedämpft. (siehe Tiefpaß-Frequenzverhalten). Das Verhalten eines RC-Glieds ist zurückzuführen auf die Tatsache daß ein Kondensator für hohe Frequenzen einen Kurzschluß und für niedrige eine Unterbrechung bedeutet.



τ=RC=1.5kΩ *33nF = 49.5µs

$t_{86\%}$ = 2* τ =99µs

max Frequenz ca. =

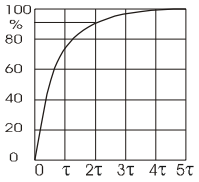
= 1/ ($t_{86\%}$ *5)

= 1/ (99µs*5)=2,02kHz

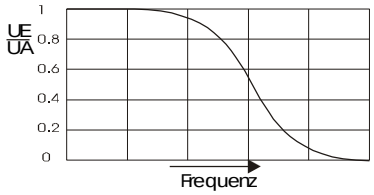
R_{40} - R_{45} = 1,5kΩ

C_1 - C_6 = 33nF

Spannungsverlauf am C beim Laden



Tiefpaß - Frequenzverhalten

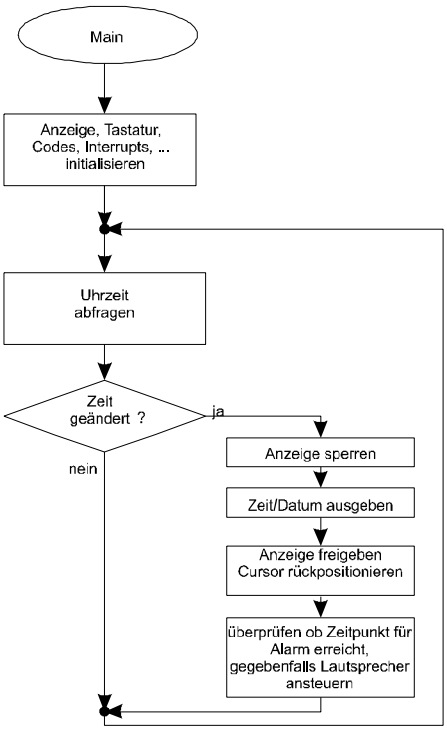


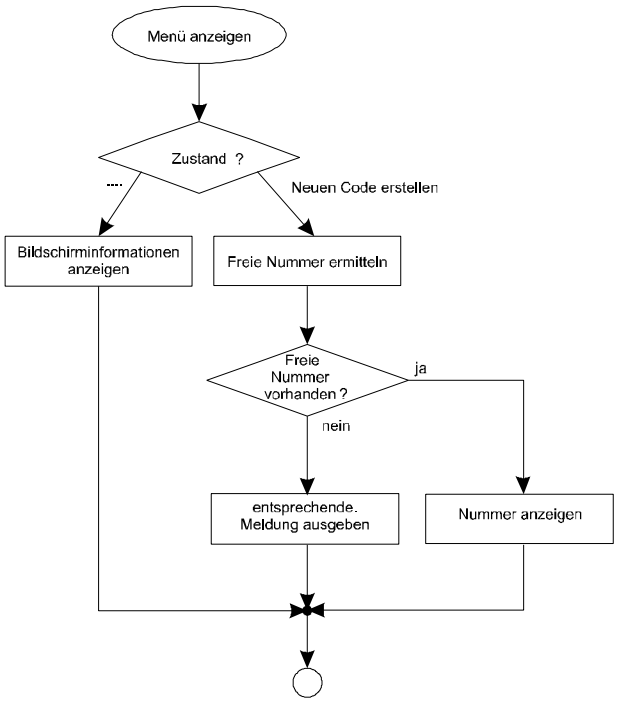
Mittels nachfolgendem Trigger werden wieder TTL-gerechte Pegel und Flanken erzeugt.

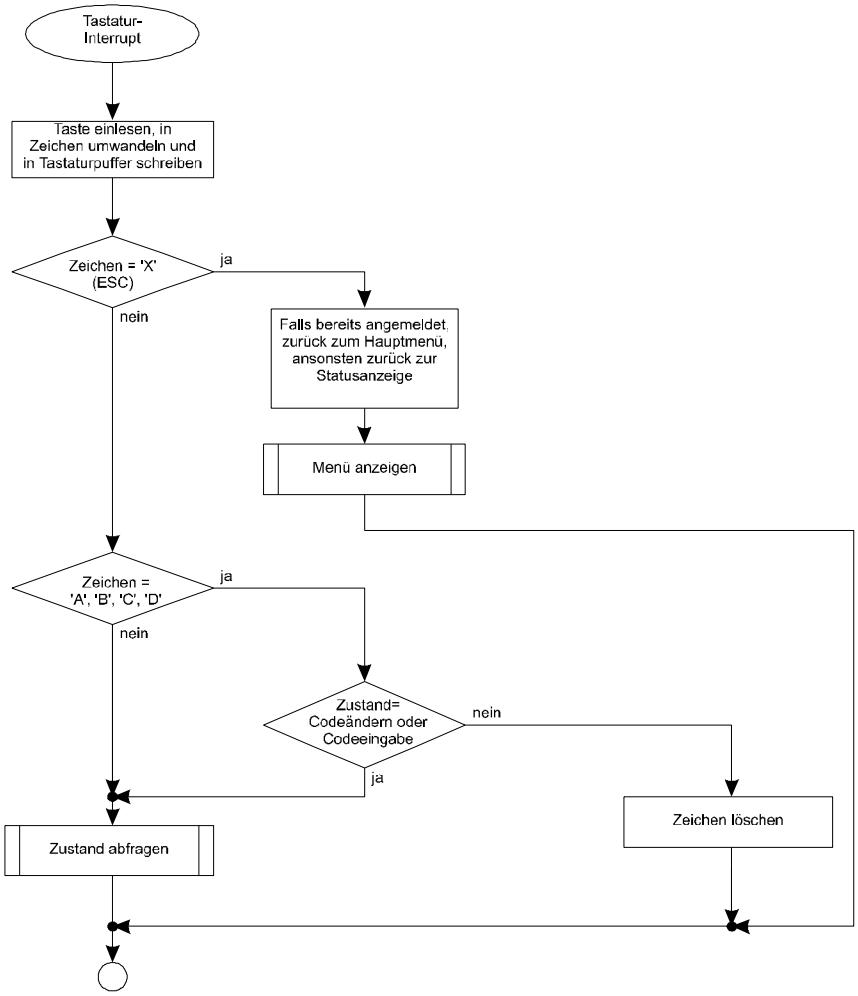
4. Das Programm

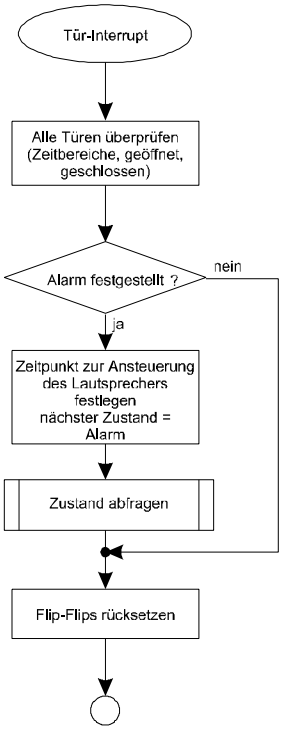
Im Prinzip haben Programme für Mikrocontroller immer einen ähnlichen Aufbau.
Sie beginnen mit Initialisierungen der externen Komponenten bzw. der Zeitgeber, Interrupts und Ports und enden in einer Endlosschleife. In meinem Programm wird in der Endlosschleife zusätzlich noch ständig der Uhrenbaustein abgefragt, und die Zeitanzeige aktualisiert. Alle Ereignisse von außen werden dem Mikrocontroller durch Interrupts mitgeteilt.
Es werden 2 Interrupts verwendet.

- Das Öffnen einer Tür bewirkt, wie bereits erwähnt, einen externen Interrupt. In der damit aufgerufenen Interruptroutione wird überprüft, ob die entsprechende Türe vom Benutzer aktiviert wurde, bzw. ob sie sich im Automatikbetrieb befindet und dementsprechend ein Alarm ausgelöst.
- Zur Abfrage der Tastatur wird ein Zeitgeber mit einer Frequenz von 100 Hz initialisiert. Nach jedem Ablaufen des Zeitgebers wird die Tastatur überprüft und, falls eine Taste gedrückt wurde, diese in einen Tastaturpuffer gestellt. Anschließend wird das Zeichen je nach Controllerzustand unterschiedlich verarbeitet. (in Routine Zustand_abfragen)

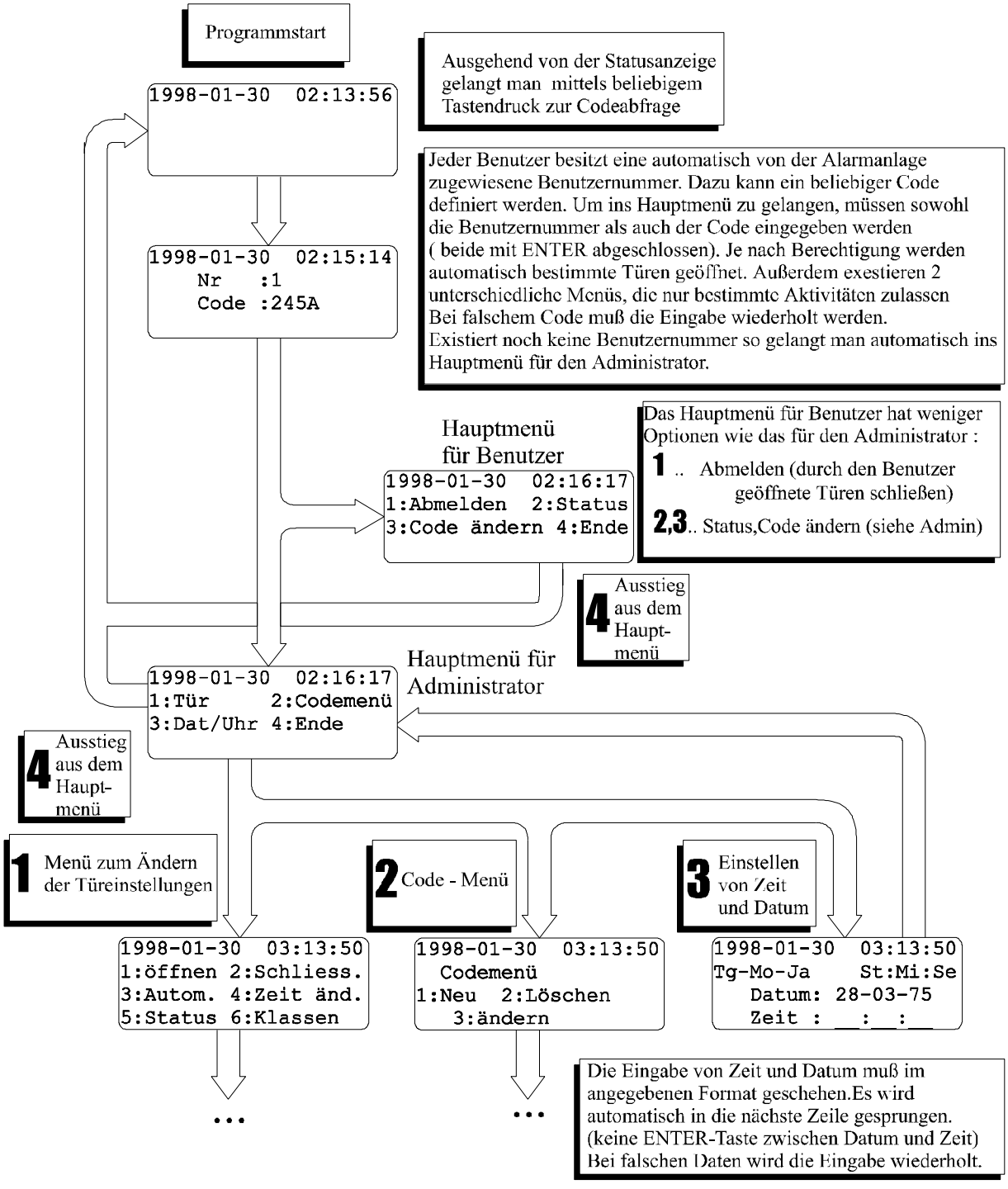

Main
Anzeige, Tastatur, Codes, Interrupts, ... initialisieren
Uhrzeit abfragen
Zeit geändert ?
ja
nein
Anzeige sperren
Zeit/Datum ausgeben
Anzeige freigeben Cursor rückpositionieren
überprüfen ob Zeitpunkt für Alarm erreicht, gegebenfalls Lautsprecher ansteuern


Menü anzeigen
Zustand ?
Neuen Code erstellen
Bildschirminformationen anzeigen
Freie Nummer ermitteln
Freie Nummer vorhanden ?
ja
nein
entsprechende. Meldung ausgeben
Nummer anzeigen

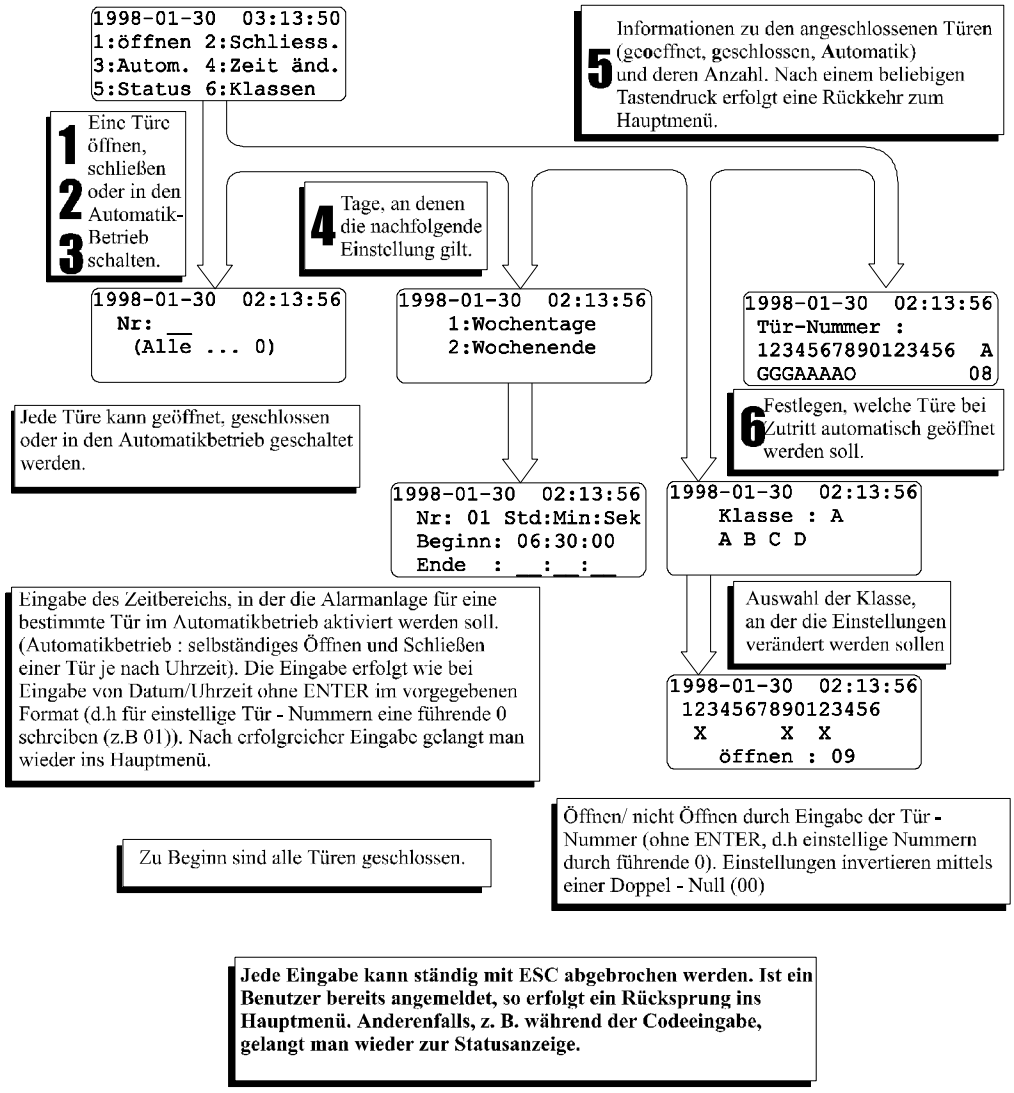

Tastatur-Interrupt
Taste einlesen, in Zeichen umwandeln und in Tastaturpuffer schreiben
Zeichen = 'X' (ESC)
ja
nein
Falls bereits angemeldet, zurück zum Hauptmenü, ansonsten zurück zur Statusanzeige
Menü anzeigen
Zeichen = 'A', 'B', 'C', 'D'
ja
nein
Zustand= Codeändern oder Codeeingabe
nein
ja
Zeichen löschen
Zustand abfragen


Tür-Interrupt
Alle Türen überprüfen (Zeitbereiche, geöffnet, geschlossen)
Alarm festgestellt ?
nein
ja
Zeitpunkt zur Ansteuerung des Lautsprechers festlegen nächster Zustand = Alarm
Zustand abfragen
Flip-Flips rücksetzen

Bedienungsanleitung

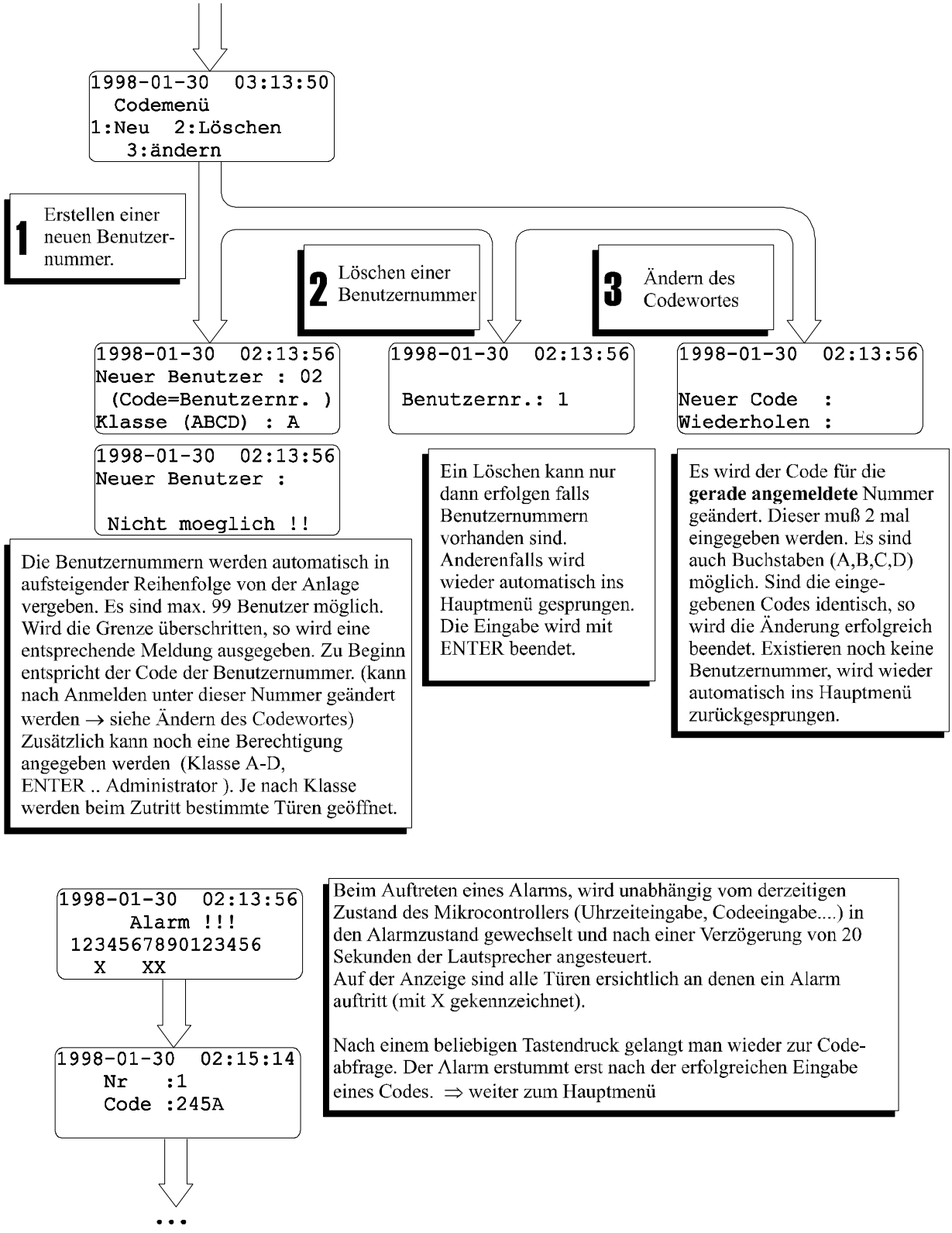


Menü zu Ändern der Türeinstellungen:



Jede Eingabe kann ständig mit ESC abgebrochen werden. Ist ein Benutzer bereits angemeldet, so erfolgt ein Rücksprung ins Hauptmenü. Anderenfalls, z. B. während der Codeeingabe, gelangt man wieder zur Statusanzeige.

Code - Menü :


1998-01-30 03:13:50
Codemenü
1:Neu 2:Löschen
3:ändern
1 Erstellen einer neuen Benutzer-nummer.
2 Löschen einer Benutzernummer
3 Ändern des Codewortes
1998-01-30 02:13:56
Neuer Benutzer : 02
(Code=Benutzernr.)
Klasse (ABCD) : A
1998-01-30 02:13:56
Neuer Benutzer :
Nicht moeglich !!
Die Benutzernummern werden automatisch in aufsteigender Reihenfolge von der Anlage vergeben. Es sind max. 99 Benutzer möglich. Wird die Grenze überschritten, so wird eine entsprechende Meldung ausgegeben. Zu Beginn entspricht der Code der Benutzernummer. (kann nach Anmelden unter dieser Nummer geändert werden → siehe Ändern des Codewortes) Zusätzlich kann noch eine Berechtigung angegeben werden (Klasse A-D, ENTER .. Administrator). Je nach Klasse werden beim Zutritt bestimmte Türen geöffnet.
1998-01-30 02:13:56
Benutzernr.: 1
Ein Löschen kann nur dann erfolgen falls Benutzernummern vorhanden sind. Anderenfalls wird wieder automatisch ins Hauptmenü gesprungen. Die Eingabe wird mit ENTER beendet.
1998-01-30 02:13:56
Neuer Code :
Wiederholen :
Es wird der Code für die **gerade angemeldete** Nummer geändert. Dieser muß 2 mal eingegeben werden. Es sind auch Buchstaben (A,B,C,D) möglich. Sind die eingegebenen Codes identisch, so wird die Änderung erfolgreich beendet. Existieren noch keine Benutzernummer, wird wieder automatisch ins Hauptmenü zurückgesprungen.
1998-01-30 02:13:56
Alarm !!!
1234567890123456
X XX
1998-01-30 02:15:14
Nr :1
Code :245A
...
Beim Auftreten eines Alarms, wird unabhängig vom derzeitigen Zustand des Mikrocontrollers (Uhrzeiteingabe, Codeeingabe....) in den Alarmzustand gewechselt und nach einer Verzögerung von 20 Sekunden der Lautsprecher angesteuert.
Auf der Anzeige sind alle Türen ersichtlich an denen ein Alarm auftritt (mit X gekennzeichnet).
Nach einem beliebigen Tastendruck gelangt man wieder zur Code-abfrage. Der Alarm erstummt erst nach der erfolgreichen Eingabe eines Codes. ⇒ weiter zum Hauptmenü

Berechnungen im Programm:

Tastaturinterrupt:

Verwendeter Timer: TIMER 4

T4CON	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0

0x040

➔ Teilungsfaktor = 16 , Timer-Betrieb , aufwärts, gestoppt,
Relode-Wert=55000

$(8*2^{\wedge}1)/_{16Mhz}*(65536-55000)$ =10,536 ms

➔ Tastaturtakt= 94,91268 Hz

T4IC	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0

0x068

Interrupt freigegeben
Interruptpriorität: ILVL=10 GLVL=0

Externer Interrupt: (Türen)

Timer 2 im Capture Mode

T2CON	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	1

0x02D

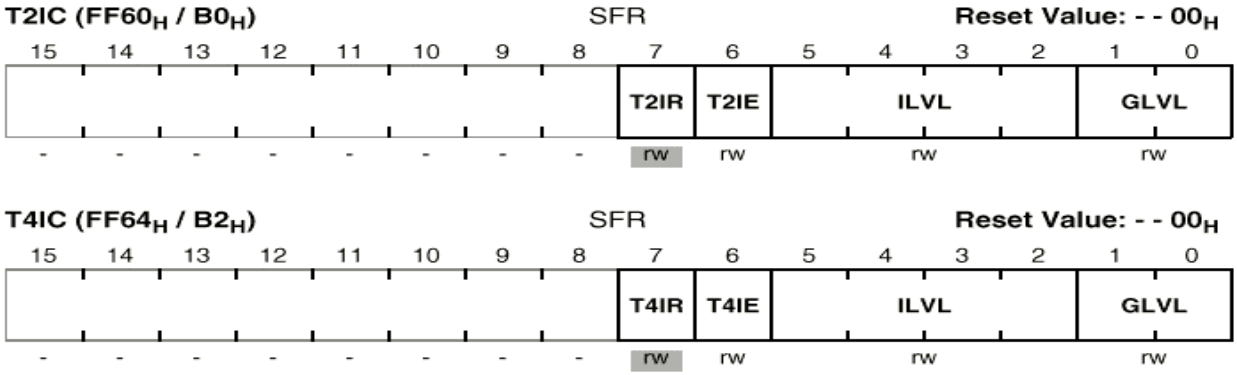
➔ pos Änderung an T2IN (P3.7) , Capture Betrieb , gestoppt

T2IC	0	0	0	0	0	0	0	0	0	1	1	0	1	1	0	0

0x06C

Interrupt freigegeben
Interruptpriorität: ILVL=11 GLVL=0

Interrupt Control - Register für Timer 2 und 4 :



Bit	Function
GLVL	**Group Level** Defines the internal order for simultaneous requests of the same priority. 3: Highest group priority 0: Lowest group priority
ILVL	**Interrupt Priority Level** Defines the priority level for the arbitration of requests. F H : Highest priority level 0 H : Lowest priority level
xxIE	**Interrupt Enable Control Bit** (individually enables/disables a specific source) ‘0’: Interrupt request is disabled ‘1’: Interrupt Request is enabled
xxIR	**Interrupt Request Flag** ‘0’: No request pending ‘1’: This source has raised an interrupt request

Timer Control - Register für Timer 2 und 4 :

T2CON ($FF40_H$ / $A0_H$) SFR **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	T2 UDE	T2UD	T2R	T2M			T2I		
-	-	-	-	-	-	-	rw	rw	rw	rw			rw		

T4CON ($FF44_H$ / $A2_H$) SFR **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	T4 UDE	T4UD	T4R	T4M			T4I		
-	-	-	-	-	-	-	rw	rw	rw	rw			rw		

Bit	Function
TxI	**Timer x Input Selection** Depends on the Operating Mode, see respective sections.
TxM	**Timer x Mode Control** (Basic Operating Mode) 0 0 0 : Timer Mode 0 0 1 : Counter Mode 0 1 0 : Gated Timer with Gate active low 0 1 1 : Gated Timer with Gate active high 1 0 0 : Reload Mode 1 0 1 : Capture Mode 1 1 X : Reserved. Do not use this combination
TxR	**Timer x Run Bit** TxR = '0': Timer / Counter x stops TxR = '1': Timer / Counter x runs
TxUD	**Timer x Up / Down Control**
TxUDE	**Timer x External Up/Down Enable**

Counter/Capture - Betrieb:

T2I / T4I	Triggering Edge for Counter Increment / Decrement
X 0 0	None. Counter Tx is disabled
0 0 1	Positive transition (rising edge) on TxIN
0 1 0	Negative transition (falling edge) on TxIN
0 1 1	Any transition (rising or falling edge) on TxIN
1 0 1	Positive transition (rising edge) of output toggle latch T3OTL
1 1 0	Negative transition (falling edge) of output toggle latch T3OTL
1 1 1	Any transition (rising or falling edge) of output toggle latch T3OTL

Timer - Betrieb :

f_{CPU} = 16 MHz	Timer Input Selection T2I / T3I / T4I							
	000B	001B	010B	011B	100B	101B	110B	111B
Prescaler factor	8	16	32	64	128	256	512	1024
Input Frequency	2 MHz	1 MHz	500 kHz	250 kHz	125 kHz	62.5 kHz	31.25 kHz	15.625 kHz
Resolution	500ns	1 µs	2 µs	4 µs	8 µs	16 µs	32 µs	64 µs
Period	33ms	66ms	131ms	262ms	524ms	1.05 s	2.1 s	4.2 s

Physikalisches Speicher-Modell:

Allgemein:

BUSCON0

0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1

0x03F

➔ 0 Wait States, 8-bit Demultiplexed

Adressbereich für Anzeige :

BUSCON1

1	1	0	0	0	1	1	0	1	1	1	1	0	0	0	0

0xC6F0

➔ 15 Wait States , 16-bit Multiplexed

ADDRSEL1

0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

0x2000

➔ BUSCON1 gilt ab Adresse 0x20:000h , 4-kB Bereich

Adressbereich für Timerbaustein :

BUSCON2 **wie bei Anzeige**

ADDRSEL2

0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0

0x3000

➔ BUSCON2 gilt ab Adresse 0x30:0000, 4 - kb Bereich

ADDRSEL1 ($FE18_H$ / $0C_H$) SFR **Reset Value: 0000_H**

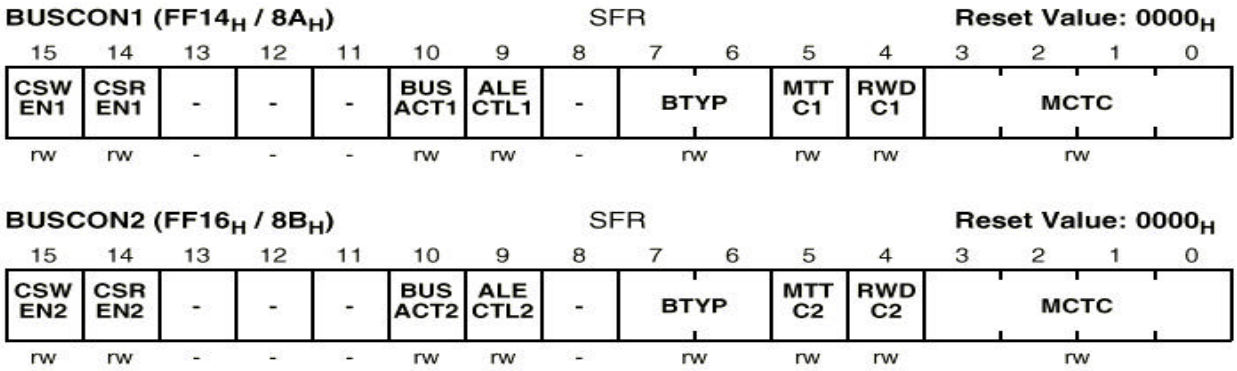
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RGSAD												RGSZ			
rw												rw			

ADDRSEL2 ($FE1A_H$ / $0D_H$) SFR **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RGSAD												RGSZ			
rw												rw			

Bit	Function
RGSZ	**Range Size Selection** Defines the size of the address area controlled by the respective BUSCONx/ ADDRSELx register pair.
RGSAD	**Range Start Address** Defines the upper bits of the start address (A23...) of the respective address area.

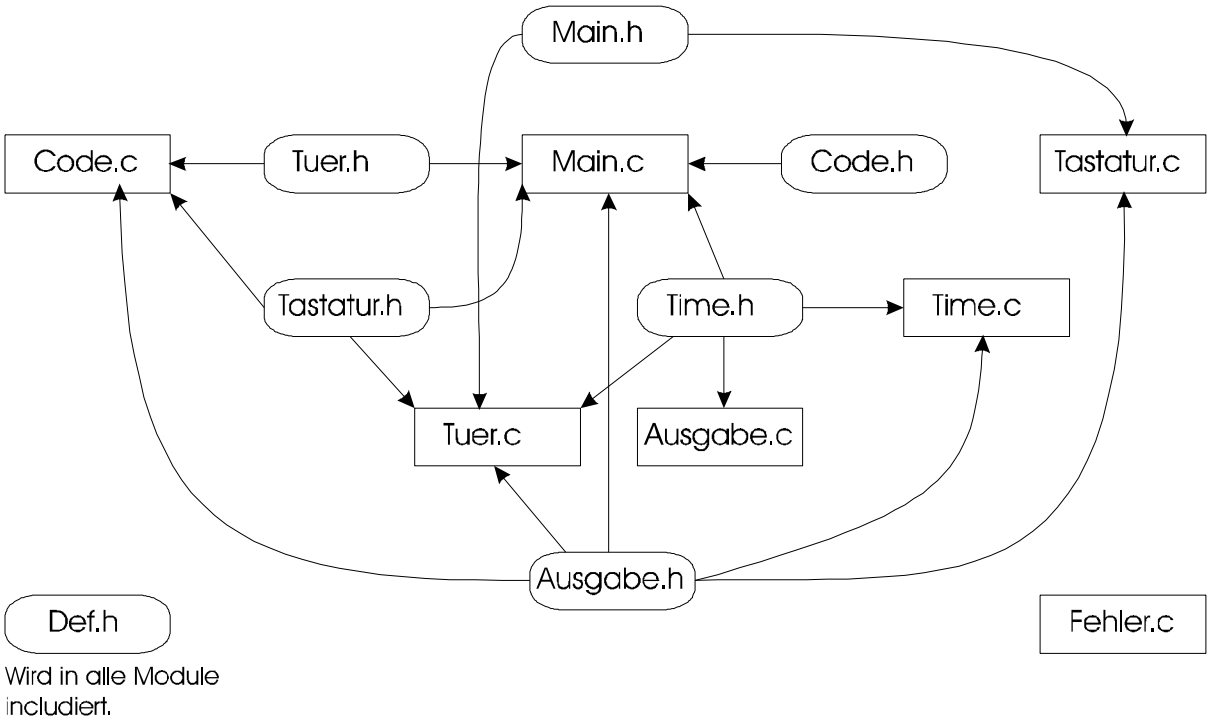
BUS - CONTROL - Register



Bit	Function
MCTC	**Memory Cycle Time Control** (Number of memory cycle time wait states) 0 0 0 0 : 15 waitstates (Number = 15 - <MCTC>) . . . 1 1 1 1 : No waitstates
RWDCx	**Read/Write Delay Control for BUSCONx** '0': With read/write delay: activate command 1 TCL after falling edge of ALE '1': No read/write delay: activate command with falling edge of ALE
MTTCx	**Memory Tristate Time Control** '0': 1 waitstate '1': No waitstate
BTYP	**External Bus Configuration** 0 0 : 8-bit Demultiplexed Bus 0 1 : 8-bit Multiplexed Bus 1 0 : 16-bit Demultiplexed Bus 1 1 : 16-bit Multiplexed Bus **Note:** For BUSCON0 BTYP is defined via PORT0 during reset.
ALECTLx	**ALE Lengthening Control** '0': Normal ALE signal '1': Lengthened ALE signal
BUSACTx	**Bus Active Control** '0': External bus disabled '1': External bus enabled (within the respective address window, see ADDRSEL)
CSRENx	**Read Chip Select Enable** '0': The CS signal is independent of the read command (RD) '1': The CS signal is generated for the duration of the read command
CSWENx	**Write Chip Select Enable** '0': The CS signal is independent of the write command (WR,WRL,WRH) '1': The CS signal is generated for the duration of the write command

Die Module:

- Das Modul Fehler.c enthält Einsprungadressen für Hard/Softwarefehler die während des Betriebs auftreten können. Diese bewirken einen Software-Reset.
- In Def.h werden alle Konstanten und Datenstrukturen definiert.
- Das Main - Modul beinhaltet den Programmstart. Hier werden die Interrupts, Ports, Timer... initialisiert und in einer nachfolgenden Schleife die Zeitanzeige aktualisiert.
  Außerdem erfolgt in diesem Modul das Darstellen der Bildschirminformationen und die Steuerung des Programmablaufes.
- Im Modul Tür.c werden Interrupts verarbeitet, die beim Öffnen einer Türe auftreten.
  Zusätzlich existieren Prozeduren zum Ändern der Tür-Einstellungen (geöffnet, geschlössen, Automatikmodus)
- Die Verwaltung der Benutzercodes erfolgt in code.c.
- Prozeduren zum Ausgeben und Einstellen der Uhrzeit bzw. des Datums befinden sich im Modul Time.c
- Das Modul Tastatur.c stellt Prozeduren zum Einlesen von Zeichen, Codes, und Zahlen und Initialisieren der Tastatur zur Verfügung.
- Die Ansteuerung des 4*20 Zeichen LCD- Display über Portleitungen erfolgt in Aus4Bit.c



Programm:

Fehler-Modul:

```
/*
    DEICHSTETTER Harald                              MAT.:9555258

    Mikrocontroller-Steuerung : ALARMANLAGE

    Fehler - Modul
    ==============
Das Modul Fehler.c enthält Einsprungadressen für Hard/Softwarefehler
die während des Betriebs auftreten können. Diese bewirken einen
Software-Reset.
                                                                 */

#include <intrins.h>
#include <string.h>
#define NMITRAP 0x02
#define STOTRAP 0x04
#define STUTRAP 0x06
#define BTRAP 0x0A


void hardware_trap__a_nmi (void) interrupt NMITRAP
{
  _trap_(0);   // SRST...Software Reset
}


void hardware_trap__a_stkof (void) interrupt STOTRAP
{
  _trap_(0);
}



void hardware_trap__a_stkuf (void) interrupt STUTRAP
{
  _trap_(0);
}



void hardware_trap_b (void) interrupt BTRAP
{
  _trap_(0);
}
```

Definition - File

```
/*
    DEICHSTETTER Harald                              MAT.:9555258

    Mikrocontroller-Steuerung : ALARMANLAGE

    Def.h
       =====
    Definition alle Konstanten und Datenstrukturen */

#include <reg161.h>

#define BYTE unsigned char

#define richtig     1      // Rückgabewerte aus Prozeduren
#define falsch      0
#define undefiniert -1
#define ausserhalb  -2

#define Enter       233    // Code für Enter am Tastenfeld
#define timer       2      // ... Timerbetrieb (Automatikmodus)
#define MAX      20        // Länge des Tastaturpuffers
#define tuer_max 16        // Maximale Anzahl der Türen
#define tuer_anz 8         // Aktuelle Anzahl der angeschlossenen Türen
#define Alarmzeitpunkt 30  // Zeit bis zum Auslösen des Alarms = 30sec
#define Codeanzahl     100 // Max. Anzahl der Benutzercodes

#define Loeschen    -1     // Controllerzustände
#define Alarm       0
#define Alarmcode   1
#define Status       2
#define Codeabfrage  3
#define Hauptmenue   4
#define Codemenue    5
#define DatumZeit    6
#define Tuermenue    7
#define Tueroeffnen  8
#define Tuerschliessen  9
#define Timeraktivieren 10
#define Tageswahl    11
#define Tueraendern1 12
#define Tueraendern2 13
#define Tuerstatus   14
#define Codeaendern  15
#define Codeloeschen 16
#define Codeneu      17
#define Klassenauswahl 18
#define KlasseA        19
#define KlasseB        20
#define KlasseC        21
#define KlasseD        22
#define Untermenue     23

        // Makro für Zugriff auf externen Speicher
#define MVAR2(object, addr)   (*((object volatile far *) (addr)))

sbit alarm = P2^10;    // Lautsprecheransteuerung
sbit tuer_offen=P3^7;  // Interrupteingang für Türen
sbit clear_FF=P2^12;   // Ausgang zum Rücksetzen der FlipFlops
sbit Zehn=P2^14;       // Ausgang für Timer (A0-Leitung)

sbit DP210=DP2^10;     // Steuerbits der Ports
sbit DP211=DP2^11;     // (zum Festlegen ob Pin als
sbit DP212=DP2^12;     //  Ausgang oder Eingang verwendet)
sbit DP214=DP2^14;

sbit TA =  P3^2;       // Tastaturansteuerung
sbit TB =  P3^3;
sbit TC =  P3^9;
sbit TD =  P3^8;
sbit TE =  P3^6;
sbit TF =  P3^13;
sbit TG =  P3^4;
sbit TH =  P3^5;

sbit DP32=DP3^2;
sbit DP33=DP3^3;
sbit DP34=DP3^4;
sbit DP35=DP3^5;
```

```
sbit T01 =  P1L^0;    // Türabfrage-Leitungen
sbit T02 =  P1L^1;
sbit T03 =  P1L^2;
sbit T04 =  P1L^3;    // Tür. Nr. 1-8
sbit T05 =  P1L^4;
sbit T06 =  P1L^5;
sbit T07 =  P1L^6;
sbit T08 =  P1L^7;
sbit T09 =  P1H^0;   // Tür. Nr. 9-11
sbit T10 =  P1H^1;
sbit T11 =  P1H^2;

sbit RSelect =   P1H^3;
sbit En   =  P2^9;
sbit Dat0 =  P1H^4;  // Display
sbit Dat1 =  P1H^5;
sbit Dat2 =  P1H^6;
sbit Dat3 =  P1H^7;
sbit DP120=DP2^9;
sbit DP121=DP1H^3;
sbit DP122=DP1H^4;
sbit DP123=DP1H^5;
sbit DP124=DP1H^6;
sbit DP125=DP1H^7;

sbit TE0 =  P1L^0;    // Zum Simulieren der Tastatur
sbit TE1 =  P1L^1;    // in der Simulationsumgebung
sbit TE2 =  P1L^2;
sbit TE3 =  P1L^3;    // Im laufenden Programm nicht mehr
sbit TE4 =  P1L^4;    // benötigt.
sbit TE5 =  P1L^5;
sbit TE6 =  P1L^6;
sbit TE7 =  P1L^7;
sbit TE8 = P0L^0;
sbit TE9 = P0L^1;
sbit A = P0L^2;
sbit B = P0L^3;
sbit C = P0L^4;
sbit D = P0L^5;
sbit ENT = P0L^6;
sbit ESC = P0L^7;

struct uhr          // Struktur für die Zeit
{
  BYTE sekunde;
  BYTE minute;
  BYTE stunde;
};
struct datum        // Struktur für das Datum
{
  BYTE tag;
  BYTE monat;
  BYTE jahr;
  BYTE woche;
};
struct eingabe      // Struktur für die Tastatureingabe
{
  BYTE puffer[MAX]; // Tastaturpuffer
  BYTE pos;         // derzeitige Position darin
};
struct tuer            // Struktur für jede Tür
{
  struct uhr begin;    // Begin und ...
  struct uhr ende;     // ... End-Zeiten an Wochentagen
  struct uhr w_begin;  // Begin und ...
  struct uhr w_ende;   // ... End-Zeiten am Wochenende
  int geoeffnet;       // Zustand der Tür (geöffnet,geschlossen,Automatikbetrieb)
  int zustand;         // Falls die Tür durch Eingabe eines Zugangscodes geöffnet
                       // wird, muß der alte Zustand zwischengespeichert werden.
  int zaehler;         // Anzeige, wieviele Benutzer diese Tür geöffnet haben
                      // Bei Zähler = 0 (kein Benutzer hält diese Tür offen)
                       // muß der alte Zustand wieder hergestellt werden.
  int alarm;           // Anzeige für Alarm
};
struct benutzer        // Struktur für Benutzercodes
{
  int belegt;          // Anzeige ob Benutzernummer belegt
  long bcode;          // Benutzercode
  int angemeldet;      // Anzeige ob Benutzer bereits angemeldet, um den Zähler bei
                       // einem 2. Anmeldeversuch nicht nocheinmal zu erhöhen.
};
```

Hauptmodul

```
/*
    DEICHSTETTER Harald                          MAT.:9555258

    Mikrocontroller-Steuerung : ALARMANLAGE

    Hauptmodul
    ==========
    Das Main - Modul beinhaltet den Programmstart. Hier werden die Interrupts, Ports, Timer...
    initialisiert und in einer nachfolgenden Schleife die Zeitanzeige aktualisiert.
    Außerdem erfolgt in diesem Modul das Darstellen der Bildschirminformationen und die
    Steuerung des Programmablaufes.*/



#include <intrins.h>
#include <stdio.h>

#include "def.h"
#include "ausgabe.h"
#include "tastatur.h"
#include "time.h"
#include "tuer.h"
#include "code.h"


int zustand,           // ... Controllerzustand
    num,               // ... freie Nummer beim Anlegen eines neuen Benutzers
    admin;             // ... Anzeige ob Administrator angemeldet
                       //     (erweitertes Hauptmenue)

extern struct eingabe eingabe;           // Tastaturpuffer


// Ausgabe der Bildschirminformationen je nach Controllerzustand.
void Menue_anzeigen(void)
{

 switch(zustand)
 {
  case(Status):
    cursor_aus();               // Cursor ausschalten und Anzeige löschen
    break;
  case(Tuerstatus) :
    cursor_aus();
    printxy(0,1," Tür-Nummer :       ");
    printxy(0,2," 1234567890123456  A");
    tuerstatus_ausgeben();
    break;
  case(Alarm):
    cursor_aus();
    printxy(0,1,"      Alarm !!!  ");
    printxy(0,2," 1234567890123456 ");
    tuerstatus_ausgeben();break;
  case(Alarmcode):
  case(Codeabfrage):
    cursor_an();                // Cursor einschalten
    printxy(3,1,"Nr   :");
    printxy(3,2,"Code :");
    eingabe_ruecksetzen();      // Hilfsvariablen zur Eingabe der Codes
    break;                      // löschen.
  case(Hauptmenue):
    cursor_aus();
    printxy(0,1,"1:Tür     2:Codemenü");
    printxy(0,2,"3:Dat/Uhr 4:Ende");
    break;
  case(Untermenue):
    cursor_aus();
    printxy(0,1,"1:Abmelden  2:Status");
    printxy(0,2,"3:Code ändern 4:Ende");
    break;
```

```
  case(Codemenue):
    cursor_aus();
    printxy(0,1,"  Codemenü:");
    printxy(0,2,"1:Neu  2:Löschen  ");
    printxy(0,3,"3:ändern          ");
    break;
  case(Codeaendern):
    cursor_an();
    printxy(0,2,"Neuer Code  :");
    printxy(0,3,"Wiederholen :");
    eingabe_ruecksetzen();       // Hilfsvariablen zur Eingabe der Codes
    break;                       // löschen.
  case(Codeloeschen):
    cursor_an();
    printxy(0,2," Benutzernr.:");
    break;
  case(Codeneu):
    cursor_aus();
    printxy(0,1,"Neuer Benutzer :    ");
    num=freie_nummer();         // Freie neue Benutzernummer ermitteln ...
    if(num)
        {
         zeige_nummer(16,1,(BYTE)num);
         printxy(0,2," (Code=Benutzernr. )"); // .. und entsprechende Meldung
          printxy(0,3,"Klasse (ABCD) :");      // ausgeben.
        }

    else
         printxy(0,3," Nicht moeglich !!");
    break;
  case(Klassenauswahl):
    cursor_aus();
    printxy(0,1,"  Klasse :  ");
    printxy(0,2,"  A B C D   ");
    break;
  case(KlasseA):
  case(KlasseB):
  case(KlasseC):
  case(KlasseD):
    cursor_an();
    printxy(0,1," 1234567890123456 ");
    printxy(0,3,"    öffnen :      ");
    klassenstatus_ausgeben();
    break;
  case(DatumZeit):
    cursor_an();
    printxy(0,1,"Tg-Mo-Ja    St:Mi:Se");
    printxy(0,2,"   Datum: __-__-__  ");
    printxy(0,3,"   Zeit : __:__:__  ");
    break;
  case(Tuermenue):
    cursor_aus();
    printxy(0,1,"1:öffnen 2:Schliess.");
    printxy(0,2,"3:Autom. 4:Zeit änd.");
    printxy(0,3,"5:Status 6:Klassem");
    break;
  case(Tueroeffnen):
  case(Tuerschliessen):
  case(Timeraktivieren):
    cursor_an();
    printxy(0,1,"  Nr: __             ");
    printxy(0,2,"   (Alle ... 0)    ");
    break;
  case(Tageswahl):
    cursor_aus();
    printxy(0,1,"    1:Wochentage");
    printxy(0,2,"    2:Wochenende");
    break;
  case(Tueraendern1):
  case(Tueraendern2):
    cursor_an();
    printxy(0,1,"  Nr: __ Std:Min:Sek");
    printxy(0,2,"  Beginn: __:__:__  ");
    printxy(0,3,"  Ende  : __:__:__  ");
    tuer_eingabe_ruecksetzen();     // Hilfsvariablen zur Eingabe der Tür-Nr.
    break;                          // und deren entsprechenden Zeiten löschen.
 }
 position();                        // Cursor an eingestellte Position setzen
}
```

```
void zurueck()
{                                 // falls Administrator angemeldet ..
  if (admin) zustand=Hauptmenue;  // .. neuer Zustand=erweitertes Hauptmenue
  else zustand=Untermenue;        // ansonsten norm. Menue
}


// Steuerung des Programmablaufes
void zustand_abfragen()
{
 int alt,   // alter Controllerzustand
     ret;   // Rückdabewert aus Prozeduren

 if(sperren())        // folgenden Code nur ausführen, falls Anzeige
                      // gerade nicht benutzt wird (z.B. von Zeit/Datum-Anzeige)
                      // Es kann sonst zu Fehlern in der Positionierung führen.

 {
   freigeben();                    // Zugriff auf die Anzeige wieder freigeben
   alt=zustand;                    // Alten Zustand merken
   switch(zustand)
    {

      case(Status):                // Falls Zustand==Status:
           taste();                // Tastenabfrage (dadurch Tastaturpuffer löschen)
           zustand=Codeabfrage;    // neuer Zustand=Codeabfrage
           setze_position(9,2);    // neue Cursorposition (9,2)
      break;                       // ( muß für Rückpositionierung nach einer
                                   //   Uhrzeitausgabe zwischengespeichert werden )

      case(Alarm):
           alt=Loeschen;           // Falls Zustand==Alarm:
           if(taste()!=undefiniert) zustand=Alarmcode;
              setze_position(9,2); // neuer Zustand = Codeabfrage bei Alarm
           break;                  // ( erst nach Tastendruck wird zur Codeabfrage
                                   //   gewechselt, (die Prozedur wird auch aus der
                                   //   Interruptroutine aufgerufen ( bei Alarm))


      case(Alarmcode):              // Falls Zustand==Codeabfrage bei Alarm
        ret=code_ueberpruefen();    // Rückgabewert der Codeabfrage ermitteln
        if(ret==richtig)
               {                            //  .... Richtig:
                   alarm_ruecksetzen();     // Alarm rücksetzen ..
                   admin=benutzerzutritt(); // .. Benutzer anmelden ..
                    zurueck();              // ... neuer Zustand=Hauptmenü
               }
        if(ret==falsch)                     //  .... Falsch:
                    zustand=Alarm;          // neuer Zustand=Alarm
        break;

      case(Codeabfrage):                        // Falls Zustand==Codeabfrage
        ret=code_ueberpruefen();                // Rückgabewert der Codeabfrage ermitteln
        if(ret==richtig)
               {
                   admin=benutzerzutritt();     // Richtig:  .. Benutzer anmelden ..
                   zurueck();                   // neuer Zustand=Hauptmenü
               }
        if(ret==falsch)  zustand=Status;        // Falsch  : neuer Zustand=Status
        break;

      case(Hauptmenue):                 // Falls Zustand==Hauptmenü (Administrator)
        ret=taste();                    // letzte Taste ermitteln
        if(ret==1) zustand=Tuermenue;   //   1: -> Türmenü
        if(ret==2) zustand=Codemenue;   //   2: -> Codemenü
        if(ret==3)  {                   //   3: -> Datum/Zeit-Menü
                    zustand=DatumZeit;
                   setze_position(10,3);
                    }
        if(ret==4) zustand=Status;      //   4: -> Ausstieg aus dem Menü zur Statusanzeige

        break;

      case(Untermenue):                 // Falls Zustand==Untermuenü (norm. Benutzer)
        ret=taste();                    // letzte Taste ermitteln
        if(ret==1) {
                     benutzeraustritt();//   1: -> Benutzer abmelden (Türen schließen)
                    zustand=Status;
                   }
        if(ret==2) zustand=Tuerstatus;  //   2: -> Status der Türen
```

```
      if(ret==3) {
                  zustand=Codeaendern;//   3: -> Code ändern
                  setze_position(13,3);
                 }
      if(ret==4) zustand=Status;      //   4: -> Ausstieg aus dem Menü zur Statusanzeige
      break;

    case(Codemenue):                 // Falls Zustand==Codemenü
      ret=taste();                   // letzte Taste ermitteln
      if(ret==1) zustand=Codeneu;    //   1: -> Neuen Code erstellen
      if(ret==2) zustand=Codeloeschen;//   2: -> Code löschen
      if(ret==3) zustand=Codeaendern; //   3: -> Code ändern
      setze_position(13,3);
      break;

   case(Codeneu):                        // Falls Zustand==Neuen Code erstellen
      ret=taste();                       // gewünschte Klasse einlesen und ..
      ret=eintrag_anlegen((BYTE)ret+(BYTE)'0'-(BYTE)'A',num); // ... Benutzer anlegen
           if(ret==richtig)
           zurueck();                    // neuer Zustand=Hauptmenü
      break;

   case(Klassenauswahl):              // Falls Zustand==Auswahl einer Klasse
      ret=taste();                    // (zur Änderung der zu öffnenden Türen)
      if(ret==(BYTE)'A'-(BYTE)'0') zustand=KlasseA;     //  A: -> Engabe für KlasseA
      if(ret==(BYTE)'B'-(BYTE)'0') zustand=KlasseB;     //  B: -> Engabe für KlasseB
      if(ret==(BYTE)'C'-(BYTE)'0') zustand=KlasseC;     //  C: -> Engabe für KlasseC
      if(ret==(BYTE)'D'-(BYTE)'0') zustand=KlasseD;     //  D: -> Engabe für KlasseD
      setze_position(13,4);
      break;

   case(KlasseA):                     // Zustand = KlasseA,B,C,D  ändern
   case(KlasseB):
   case(KlasseC):
   case(KlasseD):
      ret=klasse_aendern();           // Klasse ändern (Türen öffnen/schließen)
      if(ret==falsch)                 // .. Bei falscher Eingabe  ..
            zurueck();                // .. neuer Zustand=Hauptmenü
      if(ret==richtig)  alt=Loeschen; // ansonsten Anzeige löschen und Informationen
      break;                         // neu anzeigen

   case(Codeloeschen):               // Falls Zustand==Code löschen
      ret=code_loeschen();            // Eingabe des zu löschenden Codes
      if(ret==richtig)                //  erfolgreich       : -> Hauptmenü
             zurueck();
      if(ret==falsch)  alt=Loeschen;  //  nicht erfolgreich : Bildschirm...
      break;                          //     ...löschen und wiederholen

   case(Codeaendern):                // Falls Zustand==Code ändern
      ret=code_aendern();             // Eingabe der neuen Codenummer
      if(ret==richtig)                //  erfolgreich : -> Hauptmenü
             zurueck();
      if(ret==falsch)  alt=Loeschen;  //  nicht erfolgreich : Bildschirm...
      break;                          //     ...löschen und wiederholen

   case(DatumZeit):                  // Falls Zustand==Datum/Zeit eingeben
      ret=DatumZeit_eingeben();       // ...Eingabe..
      if(ret==richtig)
            zurueck();                //  erfolgreich : -> Hauptmenü
      if(ret==falsch)  alt=Loeschen;  //  nicht erfolgreich : Bildschirm...
      break;                          //     ...löschen und wiederholen

   case(Tuermenue):                     // Falls Zustand==Türmenü
      ret=taste();                      // letzte Taste ermitteln
      if(ret==1) zustand=Tueroeffnen;   //  1: -> Türen öffenen
      if(ret==2) zustand=Tuerschliessen; //  2: -> Türen schließen
      if(ret==3) zustand=Timeraktivieren;//  3: -> Türen auf Automatikbertrieb
      if(ret==4) zustand=Tageswahl;     //  4: -> Zeitbereiche für Türen einstellen
      if(ret==5) zustand=Tuerstatus;    //  5: -> Status der Türen
      if(ret==6) zustand=Klassenauswahl; //  6: -> Klassen ändern
      if(ret<4) setze_position(6,2);    // Bei den ersten 3 Menüpunkten neue
      break;                            // Cursorposition festlegen.

  case(Tageswahl):                      // Falls Zustand==Zeitbereiche für Türen einstellen
      ret=taste();                      // letzte Taste ermitteln
      if(ret==1) zustand=Tueraendern1;  //   1: -> Ändern für Wochentage
      if(ret==2) zustand=Tueraendern2;  //   2: -> Ändern für Wochenende
      setze_position(6,2);
      break;
```

```
    case(Tueraendern1):
    case(Tueraendern2):                    // Falls Zustand==Zeitändern für Wochenende/tage
        ret=BeginEnde_eingeben();          // Start/Ende-Zeiten eingeben
        if(ret==richtig)
              zurueck();                   //  erfolgreich : -> Hauptmenü
        if(ret==falsch)  alt=Loeschen;     //  nicht erfolgreich : Bildschirm...
        break;                             //     ...löschen und wiederholen

     case(Tueroeffnen):
     case(Tuerschliessen):
     case(Timeraktivieren):               // Falls Zustand==Tür öffenen/schließen/automatik
        ret=tuerabfrage_aendern();         //   .. Abfrage..
        if(ret==richtig)
              zurueck();                   //  erfolgreich : -> Hauptmenü
        if(ret==falsch)  alt=Loeschen;
        break;

     case(Tuerstatus):                     // Falls Zustand==Status der Türen
        taste();                           // Tastaturpuffer löschen
             zurueck();                    //  -> Hauptmenü
        break;
   }
 if (alt!=zustand) Menue_anzeigen();    // Hat sich der Zustand geändert -> Menü neu anzeigen
 }
 else eingabe.pos--;            // Falls Anzeige bereits benutzt, Menüabfrage nicht
                                // möglich, und letztes Zeichen aus Tastaturpuffer entfernen
}

//  Definieren der Ein/Ausgänge
void ports_initialisieren(void)
{
   DP210=1;                    /* Port 2.10 ..... Lautsprecherausgang */
   DP212=1;                    /* Port 2.12 ..... Ausgang für Flip-Flop */
   DP32=DP33=DP34=DP35=1;      /* Port 3.2-3.5 .. Ausgänge für Tastatur */
   DP214=1;                    /* Port 2.14 ..... Ausgang für Timer  (A0-Leitung) */
   Zehn=0      /* A0 auf 0 setzen (Einerstelle des Timerbausteines wird abgefragt)*/
   DP120=DP121=DP122=DP123=DP124=DP125=1;   /* Port 2.9, 1.11-1.15 .. Ausgänge für Display  */
}


//  Initialisieren der Interrupts
void interrupts_initialisieren(void)
{
   T2CON = 0x002D;            // Externer Interrupt an P3.7  (Tür - Interrupt)
    // Timer 2:  Capture,  positive Transition at T2IN , T2R=0
   T2IC = 0x6C;
    //  Interruptpriorität: IR=0  IE=1  ILVL=11  GLVL=0
   T4IC = 0x68;                // Tastatur : Timer 4 */
    //  Interruptpriorität: IR=0  IE=1  ILVL=10  GLVL=0
   IEN=1;                      // Interrupts zulassen
}


void main(void)
{
   zustand=Status;               // Anfangszustand = Status
   ports_initialisieren();       // Ports initialisieren
   controller_initialisieren();  // Adressbereiche und deren Timing einstellen
   ausgabe_initialisieren();
   cursor_aus();                 // Cursor ausschalten
   printxy(0,1,"  Flip Flop init.");  // Ausgabe falls sich die Flip-Flops nicht
                                      // rücksetzen lassen
                                      //  (MC bleibt bei nächten Befehl stehen)
   loesche_FF();                 // Flip-Flops der Türen rücksetzen
   printxy(0,1,"  Tastatur init.");
   tastatur_initialisieren();    // Tastatur initialisieren
   tuer_initialisieren();
   printxy(0,1,"  Türen        ");
   code_initialisieren();        // Codes       -- "" --
   printxy(0,1,"  Code         ");
   klassen_initialisieren();
   printxy(0,1,"  Interrupt     ");
   interrupts_initialisieren();  // Interrupts  -- "" --
   freigeben();                  // Anzeige freigeben
   cursor_aus();                 // Cursor ausschalten (Bildschirm löschen)
   while(1)
     {
               uhr_abfragen();         // Uhr ständig abfragen und anzeigen
               alarm_aktivieren();     // falls Alarmzeitpunkt erreicht, diesen aktivieren
     }
}
```

```
/*
    DEICHSTETTER Harald                          MAT.:9555258

    Mikrocontroller-Steuerung : ALARMANLAGE

    Main.h                                                    */


void zustand_abfragen(void);
   // Ausgabe der Bildschirminformationen je nach Controllerzustand.
void Menue_anzeigen(void);
   // Steuerung des Programmablaufes
void zurueck(void);
   // Rückkehr ins erweitertes Hauptmenue falls Administrator angemeldet,
   // ansonsten ins norm. Menue
```

Türen- Modul

```
/*
    DEICHSTETTER Harald                              MAT.:9555258

    Mikrocontroller-Steuerung : ALARMANLAGE

    Türen - Modul
    ================
       Im Modul Tür.c werden Interrupts verarbeitet, die beim Öffnen einer
       Tür auftreten. Zusätzlich existieren Prozeduren zum Ändern der
       Tür-Einstellungen (geöffnet, geschlössen, Automatikmodus)
                                                                      */

#include "def.h"
#include "main.h"
#include "ausgabe.h"
#include "time.h"
#include "tastatur.h"
#include <stdio.h>
#include <stdlib.h>

extern struct eingabe eingabe;   // Tastaturpuffer
extern struct uhr uhr;           // Uhrzeit zur Steuerung der Verzögerung des Alarm
extern int zustand;              // Controllerzustand

int klasse[5];                   // Klassen (A,B,C,D,Admin)
struct tuer tuer[16];            // Türen
int  tuer_nr;                    // Hilfsvariable
int alarm_begin;                 // Alarmstartzeit


//  2^y
// Eigene POW-Routine, da die vorgefertigte Funktion zu viel Speicher im
// Controller benötigt, und in diesem Fall Geschwindigkeit keine Rolle spielt.
int mypow(int y)
{
 int i,j;
 if (y==0) return 1;
 i=2;
 for(j=1;j<y;j++)
    i=i*2;
 return i;
}


// Status der Türen ausgeben
void tuerstatus_ausgeben(void)
{
 int i;        // Hilfsvariablen
 BYTE txt[20];

 txt[15]=(BYTE)' ';
 txt[16]=(BYTE)' ';
 txt[17]=(BYTE)' ';
 txt[18]=(BYTE)' ';
 txt[19]=(BYTE)'\0';

 for(i=0;i<tuer_max;i++)
   {
    if(zustand==Alarm)
       {           // Bei Zustand=Alarm ...
                   // ... Türen anzeigen an denen Alarm auftritt
        if(tuer[i].alarm==richtig) txt[i]=(BYTE)'X';
        else txt[i]=(BYTE)' ';
       }
                   // Sonst Türmodus ausgeben (O..Geöffnet, G..Geschlossen, A..Automatik )
    else if(tuer[i].geoeffnet==richtig) txt[i]=(BYTE)'O';
    else if(tuer[i].geoeffnet==falsch)  txt[i]=(BYTE)'G';
    else if(tuer[i].geoeffnet==timer) txt[i]=(BYTE)'A';
    if(i>=tuer_anz) txt[i]=(BYTE)' ';  // .. für alle nicht angeschlossenen Türen
   }

 if (zustand!=Alarm)
   {
    txt[17]=(BYTE)(tuer_anz/10+'0');   // Anzahl der angeschlossenen Türen ausgeben
    txt[18]=(BYTE)(tuer_anz-(tuer_anz/10)*10+'0');
   }
 printxy(1,3,txt);
}
```

```
//  Start und Eindzeiten der Tür zuweisen
//  Eingabestrukturen korrekt    -> richtig
//  ------- "" ----   falsch     -> falsch
int tuer_setzen(struct datum begin,struct uhr ende)
                        // Aufgrund von Vereinfachungen in der Eingabe
{                       // wurde für die Startzeit die datum-Struktur
 if(ende.sekunde<60)    // verwendet
     if(ende.minute<60)
        if(ende.stunde<24)              // Endzeitpunkt überprüfen ...
                {
                if(begin.jahr<60)               // Startzeitpunkt überprüfen ...
                 if(begin.monat<60)
                 if(begin.tag<24)
                      {
                         if(zustand==Tueraendern1)  // ... und zuweisen.
                           {                        // (für Wochentage)
                             tuer[tuer_nr-1].ende=ende;
                            tuer[tuer_nr-1].begin.stunde=begin.tag;
                             tuer[tuer_nr-1].begin.minute=begin.monat;
                             tuer[tuer_nr-1].begin.sekunde=begin.jahr;
                           }
                         else
                           {                        // (für Wochenende)
                             tuer[tuer_nr-1].w_ende=ende;
                            tuer[tuer_nr-1].w_begin.stunde=begin.tag;
                             tuer[tuer_nr-1].w_begin.minute=begin.monat;
                             tuer[tuer_nr-1].w_begin.sekunde=begin.jahr;
                           }
                         return richtig;
                      }
                 }

 return falsch;
}


// Hilfsvariablen für Türeingabe löschen
void tuer_eingabe_ruecksetzen(void)
{
 tuer_nr=undefiniert;
}



// Start und Endzeitpunkte für eine bestimmte Tür eingeben
// Eingabe noch nicht komplett  -> undefiniert
// -- " -- korrekt              -> richtig
// -- " -- nicht korrekt        -> falsch
int BeginEnde_eingeben(void)
{
 struct datum begin;
 struct uhr ende;

 if (tuer_nr==undefiniert)
      {                     // Falls Tür-Nummer noch nicht eingegeben wurde..
       tuer_nr=eingabe_in_nummer(5,2,tuer_anz);   // ... diese eingeben
       if(tuer_nr==ausserhalb) return falsch;
                            // Falls Eingabe zu groß (>tuer_anz) ... Aussteigen
       if(eingabe.pos==0) setze_position(10,3);
                           // Bei fertiger Eingabe Cursor nachpositionieren
      }

 else if (eingabe.pos==12)  // Wurde bereits Begin/Endzeit eingegeben
     {                      // ((std,min,sec)=6*2=12
     eingabe_in_datum(&begin); // Tastaturpuffer in Begin und Endzeit
     eingabe_in_zeit(&ende);   // umwandeln und ...
     taste();                  // .. löschen.
     if (tuer_setzen(begin,ende))  // Zeiten der Tür zuweisen.
            return richtig;
     setze_position(6,2);      // Falls Zuweisung nicht erfolgreich..
     return falsch;            // Cursorposition wieder zum Begin setzen
     }

 else Zeit_Eingabe_anzeigen(); // Anderenfalls Eingabe ausgeben.
 return undefiniert;  // Bei noch unvollständiger Eingabe -> undefiniert zurückgeben
}
```

```
// Einer Tür(nr) einen neuen Zustand zuweisen.
void aendern(int nr)
{
 switch(zustand)
      {      // Je nach Controllerzustand(Menüeintrag)
       case(Tueroeffnen):
           tuer[nr-1].geoeffnet=richtig;break; // Tür öffnen
       case(Tuerschliessen):
           tuer[nr-1].geoeffnet=falsch;break;  // schließen
       case(Timeraktivieren):
            tuer[nr-1].geoeffnet=timer;break;   // Automatik
      }
}


// Türmodus einer/aller Türen ändern
// Eingabe noch nicht komplett  -> undefiniert
// Eingabe korrekt              -> richtig
// -- " -- nicht korrekt        -> falsch
int tuerabfrage_aendern(void)
{
 int i;
 int nr;
 nr=eingabe_in_nummer(5,2,tuer_anz); // Türnummer eingeben
 if(nr==ausserhalb) return falsch;   // Falls Eingabe zu groß (>tuer_anz) ...
                                    //    ... Aussteigen
 if(nr!=undefiniert)
 {                                   // wurde erfolgreich eingegeben....
   if(nr==0)                         // ... Eingabe==0 alle Türen ändern
      for(i=1;i<=tuer_anz;i++)
              aendern(i);
   else aendern(nr);                 //     ansonsten jeweilige Türe ändern
   return richtig;
   }
 return undefiniert;    // Bei noch unvollständiger Eingabe -> undefiniert zurückgeben
}


// Türen auf Starteinstellungen setzen
void tuer_initialisieren(void)
{
 int i;
 for(i=0;i<tuer_max;i++)
    {
    tuer[i].geoeffnet=falsch;   // Alle Türen geschlossen
    tuer[i].alarm=falsch;       // kein Alarm aufgetreten
    tuer[i].zustand=undefiniert;// letzter Zustand undefiniert
    tuer[i].zaehler=0;          // von keinem Benutzer geöffnet
    }
 alarm_begin=undefiniert;       // Alarmbeginn - Zeitpunkt undefiniert
}


// Überprüft ob bei einer Tür(nr) ein Alarm auftritt
// Alarm       -> richtig
// kein Alarm  -> falsch
int tuer_ueberpruefen(int nr)

{
 int zahl, port1=0,// Hilfsvariablen für Portabfrage
 in_zeit;         // Richtig falls derzeitige Zeit innerhalb Start und Endzeitpunkt

         // Derzeitige Zeit mit der der Tür zugeordneten Start und Endzeit vergleichen
 if(wochenende()) in_zeit=zeit_vergleichen(tuer[nr].w_begin,tuer[nr].w_ende);
 else in_zeit=zeit_vergleichen(tuer[nr].begin,tuer[nr].ende);

 if (  (in_zeit&&(tuer[nr].geoeffnet==timer))
       ||(tuer[nr].geoeffnet==falsch))
   {     // Falls die Türabfrage auf Automatikbetrieb und die derzeitige Zeit sich
         // innerhalb den vorgegebenen Grenzen befindet oder die Türe ist geschlossen ...
    zahl=mypow(nr);              // Anschluß der Tür nr am Port1 berechnen
    port1=(int)P1L+256*(int)P1H; // ... Port1 abfragen..
                                 // ... und vergleichen
    if(zahl&port1) return richtig; // ... ist Tür offen -> Alarm
   }
 return falsch;
}
```

```
// Flip-Flops löschen
void loesche_FF(void)
{
 clear_FF=falsch;               // clear_FF ..LOW-AKTIV
 while(tuer_offen)              // solange FF rücksetzen bis alle Türen als
       printf("FF-ruecksetzen"); // geschlossen erkannt werden.
 clear_FF=richtig;
}



// Interruptroutine bei Öffnen einer Tür
void tuer_interrupt(void) interrupt 0x22 using regbank_003
{
 int i;
 for(i=0;i<tuer_anz;i++)
    if(tuer_ueberpruefen(i))  // Alle Türen überprüfen
     {
       tuer[i].alarm=richtig; // gegebenfalls Alarm setzen
       zustand=Alarm;         // Controllerzustand auf Alarm setzen.
     }

 if (zustand==Alarm)           // Bei aufgetretenem Alarm ...
     {
      if (alarm_begin==undefiniert) alarm_begin=(uhr.sekunde+Alarmzeitpunkt)%60;
             // tatsächlichen Alarmzeitpunkt berechnen (+ definierte sec)
      printf("\nAlarm !!\n");
      taste();                  // Tastaturpuffer löschen
      freigeben();              // Anzeige freigeben (Alarm soll auch bei gerade
                                // benutzter Anzeige ausgeführt werden)
      zustand_abfragen();       // in Alarmzustand wechseln
     }
 loesche_FF();                 // Flip-Flops wieder rücksetzen
}


// Alarm an allen Türen zurücksetzen
// und Ansteuerung für Lautsprecher aufheben
void alarm_ruecksetzen(void)

{
 int i;
 for(i=0;i<tuer_max;i++)
    tuer[i].alarm=falsch; // kein Alarm
 alarm=falsch;            // Ansteuerung für Lautsprecher aufheben.
 alarm_begin=undefiniert; // Alarmzeitpunkt undefiniert
}


// Benutzerklassen initialisieren
void klassen_initialisieren(void)
{
 int i;
 for(i=0;i<4;i++)
        klasse[i]=0; // bei Klasse A,B,C,D wird keine Tür zu Beginn geöffnet
 klasse[4]=0;  // für Admin öffnet sich ebenfalls keine Türe
}


// Einstellungen für Klasse je nach Controllerzustand ändern
int klasse_aendern(void)
{
 int nr;

 nr=eingabe_in_nummer(12,4,tuer_anz);  // Türnummer eingeben

 if(nr!=undefiniert)
   {
    if(nr!=ausserhalb)           // falls erfolgreich eingegeben....
       {
        if(nr==0)                // Bei Eingabe==0 alle Türen invertieren
             klasse[zustand-KlasseA]=klasse[zustand-KlasseA]^mypow(tuer_anz)-1;
        else klasse[zustand-KlasseA]=klasse[zustand-KlasseA]^mypow(nr-1);
                                 // ansonsten jeweilige Türe ändern
                    //  ( die zu ändernde Klasse = aktueller Zustand-KlasseA )
       }
    setze_position(13,4);        // Rückpositionieren
    return richtig;
   }
 return undefiniert;   // Bei noch unvollständiger Eingabe -> undefiniert zurückgeben
}
```

```
// Ausgabe der Türen die für eine bestimmte Klasse automatisch geöffnet werden
void klassenstatus_ausgeben(void)
{
 int i;
 BYTE txt[17];

 txt[16]=(BYTE)'\0';
 for(i=0;i<tuer_max;i++)
       {
       if (klasse[zustand-KlasseA]&mypow(i)) txt[i]=(BYTE)'X'; //... geöffnet
       else txt[i]=(BYTE)' ';                                  // nicht geöffnet
       }

 printxy(1,2,txt);
}


// Türen für eine bestimmte Klasse(k) öffnen
void tueren_oeffnen(int k)
{
 int i;
 for(i=0;i<tuer_max;i++)
   if (klasse[k]&mypow(i))             // Abfrage ob Tür geöffnet werden soll..
              {
              tuer[i].zaehler++;       // Zähler für die zu öffnende Tür erhöhen
              if (tuer[i].zaehler==1)  // falls Türe das erste Mal geöffnet wird ...
                           {
                           tuer[i].zustand=tuer[i].geoeffnet; //... alten Zustand merken
                           tuer[i].geoeffnet=richtig;         // und Tür öffnen
                           }
              }
}


// Türen für eine bestimmte Klasse(k) schließen
void tueren_schlieszen(int k)
{
 int i;
 for(i=0;i<tuer_max;i++)
   if (klasse[k]&mypow(i))       // Abfrage ob Tür geschlossen werden soll..
                      {
                      tuer[i].zaehler--;      // Zähler für die zu schließende Tür vermindern
                      if (tuer[i].zaehler==0) // Falls niemand mehr die Tür offen hält
                           {
                            tuer[i].geoeffnet=tuer[i].zustand; // Alten Zustand wieder herstellen
                           tuer[i].zustand=undefiniert;       // Merker auf undefiniert setzen
                           }
                      }

}


// Überprüft ob der Alarmzeitpunkt erreicht wurde, und löst den Alarm gegebenfalls aus.
void alarm_aktivieren(void)
{
 if (alarm_begin!=undefiniert) printf("Counter : %i ",uhr.sekunde-alarm_begin);
 if (uhr.sekunde==alarm_begin)  // .. Alarmzeitpunkt erreicht ?
       {
         alarm_begin=undefiniert; // Beginnzeitpunkt wieder undefiniert
        alarm=richtig;           // Lautsprecher ansteuern
       }
}
```

```
/*
    DEICHSTETTER Harald                          MAT.:9555258

    Mikrocontroller-Steuerung : ALARMANLAGE

    Tuer.h
         */



void tuerstatus_ausgeben(void);
 // Status der Türen ausgeben
int BeginEnde_eingeben(void);
 // Start und Endzeitpunkt für eine bestimmte Tür eingeben
 // Eingabe noch nicht komplett  -> undefiniert
 // -- " -- korrekt              -> richtig
 // -- " -- nicht korrekt        -> falsch
int tuerabfrage_aendern(void);
 // Türmodus einer/aller Türen ändern
 // Eingabe noch nicht komplett  -> undefiniert
 // Eingabe korrekt              -> richtig
 // -- " -- nicht korrekt        -> falsch
void tuer_initialisieren(void);
 // Türen auf Starteinstellungen setzen
void alarm_ruecksetzen(void);
 // Alarm an allen Türen zurücksetzen
 // und Ansteuerung für Lautsprecher aufheben
void loesche_FF(void);
 // Flip-Flops rücksetzen
void tuer_eingabe_ruecksetzen(void);
 // Hilfsvariablen für Türeingabe löschen
int klasse_aendern(void);
 // Einstellungen für Klasse je nach Controllerzustand ändern
 // Eingabe noch nicht komplett  -> undefiniert
 // Eingabe korrekt              -> richtig
void klassenstatus_ausgeben(void);
 // Ausgabe der Türen die für eine bestimmte Klasse automatisch geöffnet werden sollen
void klassen_initialisieren(void);
 // Benutzerklassen initialisieren
void tueren_schlieszen(int k);
 // Türen für eine bestimmte Klasse(k) schließen
void tueren_oeffnen(int k);
 // Türen für eine bestimmte Klasse(k) öffnen
void alarm_aktivieren(void);
 // Überprüft ob der Alarmzeitpunkt erreicht wurde, und löst den Alarm gegebenfalls aus.
```

Code - Modul

```
/*
    DEICHSTETTER Harald                              MAT.:9555258

    Mikrocontroller-Steuerung : ALARMANLAGE

    Code - Modul
    ================
    Verwaltung der Benutzercodes

    ( Wurde auch mit dynamischer Datenstruktur programmiert.
      Diese funktioniert jedoch nur am Simulator und nicht
      am Controller)
                                                              */

#include<stdlib.h>
#include<string.h>
#include"def.h"
#include"tastatur.h"
#include"ausgabe.h"
#include"Tuer.h"

int nummer;         // Hilfsvariable
long code1;         // Hilfsvariable
int nr;             // aktuell angemeldete Benutzernummer

extern struct eingabe eingabe;          // Tastaturpuffer
struct benutzer benutzer[Codeanzahl];   // Benutzercodes
extern int admin;   // ... Anzeige ob Administrator angemeldet


// Überprüft ob Benutzercodes vorhanden
//  vorhanden        -> falsch
//  nicht vorhanden  -> richtig
int codes_leer(void)
{
 int i,leer=richtig;
 i=-1;
 do
   {
   i=i+1;                              // Falls Benutzer bereits belegt ->
   if(benutzer[i].belegt) leer=falsch; // leer = falsch
   }
 while ((i<Codeanzahl-1) && leer);
 return (leer);
}


// Liest und Überprüft die eingegebenen Codes
//   Eingabe noch nicht komplett  -> undefiniert
//   Code korrekt                 -> richtig
//    -"- nicht korrekt           -> falsch
int code_ueberpruefen(void)
{
 long code=undefiniert;
 if (codes_leer())
       {
         taste();        // Falls noch keine Codes gespeichert
        return richtig; // -> richtig (Automatisch ins Hauptmenü)
       }
                        // Wurde noch keine Benutzernummer eingegeben..
 if (nummer==undefiniert) nummer=(int)code_eingabe(9,1); // .. diese eingeben

 else code=code_eingabe(9,2); // Ansonsten Code eingeben

 if(code!=undefiniert) // Wurden bereits Nummer und Code eingegeben ..
   {
    nr=nummer;         // ... Benutzernummer merken (diese ist ab nun angemeldet) ..
    taste();           // ... Tastaturpuffer löschen ...
    return (nr>0)&&(benutzer[nr-1].bcode==code)&&(benutzer[nr-1].belegt);
 }                    // ... Codes vergleichen und Ergebnis zurückgeben
 return undefiniert;   // Bei noch unvollständiger Eingabe -> undefiniert
}
```

```
// Hilfsvariablen für Codeeingabe löschen
void eingabe_ruecksetzen(void)
{
 nummer=undefiniert;
 code1=undefiniert;
}


// Codes auf Starteinstellung setzen (löschen)
void code_initialisieren(void)
{
 int i;
 nr=undefiniert;         // kein Benutzer angemeldet
 for (i=0;i<Codeanzahl;i++)
    benutzer[i].belegt=falsch;    //  alle Benutzer nicht belegt
    benutzer[i].angemeldet=falsch;//  und nicht angemeldet
}



// Benutzer (nummer) neuen Code(bcode) zuweisen
//  Benutzer noch nicht belegt -> falsch
//  ansonsten                  -> richtig
int eintrag_aendern(int nummer,long bcode)
{
  if (!benutzer[nummer-1].belegt) return falsch; // nicht belegt -> falsch
  benutzer[nummer-1].bcode=bcode;   // Anderenfalls neue Benutzernummer speichern
  return richtig;
}


// Benutzercode ändern
//  Eingabe noch nicht komplett          -> undefiniert
//  2 eingegebene Codes nicht identisch  -> falsch
//  Zuweisung erfolgreich                -> richtig
int code_aendern(void)
{
 long code2=undefiniert;

 if (nr==undefiniert) return richtig; // Falls kein Benutzer angemeldet -> Rückkehr
 if(code1==undefiniert)               // Neuen Code eingeben....
      code1=code_eingabe(13,2);
 else code2=code_eingabe(13,3);       // .. danach wiederholen
 if ((code1!=undefiniert)&&(code2!=undefiniert))
       {                             // sind beide Codes eingegeben...
        setze_position(13,3);        // ... Cursorposition setzen..
           if(code1==code2) return eintrag_aendern(nr,code1);
        return falsch;               // .. falls Codes identisch
       }                             // .. Eintrag ändern..
 return undefiniert;
}


// Löscht Benutzer mit der Nummer nummer
// Falls nicht belegt -> falsch
// ansonsten          -> richtig
int eintrag_loeschen(int nummer)
{
 if(!benutzer[nummer-1].belegt) return falsch;
 benutzer[nummer-1].belegt=falsch;      // löschen
 return richtig;
}


//  Benutzercode löschen
//  Eingabe noch nicht komplett          -> undefiniert
//  Benutzer existiert nicht             -> falsch
//  Löschvorgang erfolgreich             -> richtig
int code_loeschen(void)
{
 int nummer;
 if (nr==undefiniert) return richtig; // Falls kein Benutzer angemeldet -> Rückkehr
 nummer=(int) code_eingabe(12,2);     // Ansonsten Benutzernummer eingeben
 if(nummer>=Codeanzahl) return falsch;
 if(nummer!=undefiniert)              // Bei erfolgreicher Eingabe ..
       return eintrag_loeschen(nummer); // .. löschen
 return undefiniert;    // Bei noch unvollständiger Eingabe -> undefiniert
}
```

```
// Ermittelt nächste freie Benutzernummer
//  keine freie Nummer mehr vorhanden   -> falsch
//  ansonsten                           -> Nummer
int freie_nummer(void)
{
 unsigned char min=0;
 while (benutzer[min].belegt && (min<Codeanzahl))
        min++;
 if (min==Codeanzahl) return falsch; // Falls Benutzerliste bereits voll ->falsch
 return min+1;           // Ansonsten Nummer zurückgeben.
}                        // ( +1 da Nummern bei 1 starten, das Array aber bei 0 )



// Neuen Benutzer mit Nummer nummer und Klasse k anlegen
//  Benutzer bereits belegt, oder falsche Klasse -> falsch
//  ansonsten                                    -> richtig
int eintrag_anlegen(int k,int nummer)
{
 if(benutzer[nummer-1].belegt) return falsch;       // Falls Benutzer schon belegt -> falsch
 if ((k<4)&&(k>=0))  benutzer[nummer-1].belegt=k+1; // Anderenfalls Klasse zuweisen (A,B,C,D)
 else if (k==Enter)  benutzer[nummer-1].belegt=5;   // oder Administrator falls bei Eingabe
                                                    // Enter gedrückt wurde
 else return falsch;
 benutzer[nummer-1].bcode=(long)nummer;            // und als Code die Benutzernummer speichern
 return richtig;
}


// Öffnen der jeweiligen Türen nach Anmelden eines Benutzers
//    Administrator angemeldet   -> richtig
//    norm. Benutzer -- "" ---   -> falsch
int benutzerzutritt(void)
{
 if ((nr!=undefiniert)&& (benutzer[nr-1].belegt-1!=4)) // Falls Klasse = A,B,C oder D ...
      {
      if (!benutzer[nr-1].angemeldet)
                {                                        // ... und Benutzer noch nicht angemeldet,
                 tueren_oeffnen(benutzer[nr-1].belegt-1); // ... Türen öffnen...
                 benutzer[nr-1].angemeldet=richtig;       // ... und anmelden
                }
      return falsch;
     }
 return richtig; // Beim erstmaligen Einstieg ins Programm (nr=undef.) oder Anmelden
                 // eines Admin. -> richtig
}


// Schließen der jeweiligen Türen nach Abmelden eines Benutzers
void benutzeraustritt(void)
{
 if ((nr!=undefiniert)&& benutzer[nr-1].angemeldet)
              {                                                  // falls Benutzer angemeldet
                     tueren_schlieszen(benutzer[nr-1].belegt-1); // ... Türen schließen ...
                     benutzer[nr-1].angemeldet=falsch;           // ... und abmelden
              }
}
```

```
/*
    DEICHSTETTER Harald                          MAT.:9555258

    Mikrocontroller-Steuerung : ALARMANLAGE

    Code.h                                                    */

int code_ueberpruefen(void);
 // Liest und Überprüft die eingegebenen Codes
 // Eingabe noch nicht komplett  -> undefiniert
 // Code korrekt              -> richtig
 //  -"- nicht korrekt        -> falsch
void code_initialisieren(void);
 // Codes auf Starteinstellung setzen (löschen)
int eintrag_anlegen(int nummer,int klasse);
 // Neuen Benutzer anlegen (neuer Code = Benutzernummer)
 //  Benutzernummer bereits belegt oder falsche Klasse -> falsch
 //  ansonsten                                        -> richtig
int code_aendern(void);
 // Benutzercode ändern
 //  Eingabe noch nicht komplett          -> undefiniert
 //  2 eingegebene Codes nicht identisch  -> falsch
 //  Zuweisung erfolgreich                -> richtig
int code_loeschen(void);
 // Benutzercode löschen
 //  Eingabe noch nicht komplett          -> undefiniert
 //  Benutzer existiert nicht             -> falsch
 //  Löschvorgang erfolgreich             -> richtig
int freie_nummer(void);
 // Ermittelt nächste freie Benutzernummer
 //  keine freie Nummer vorhanden   -> falsch
 //  ansonsten                      -> Nummer
void eingabe_ruecksetzen(void);
 // Hilfsvariablen für Codeeingabe löschen
int benutzerzutritt(void);
 // Öffnen der jeweiligen Türen nach Anmelden eines Benutzers
 //    Administrator angemeldet   -> richtig
 //    norm. Benutzer -- "" ---   -> falsch
void benutzeraustritt(void);
 // Schließen der jeweiligen Türen nach Abmelden eines Benutzers
```

Zeit/Datum - Modul

```
/*
    DEICHSTETTER Harald                              MAT.:9555258

    Mikrocontroller-Steuerung : ALARMANLAGE

    Zeit/Datum - Modul
    ==================
    Prozeduren zum Einstellen und Ausgeben der Uhrzeit bzw. des Datums    */

#include "def.h"
#include "ausgabe.h"
#include "time.h"
#include <absacc.h>
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>

#define sec   MVAR(unsigned char,0x300000) /* Datenregister des Timerbausteins*/
#define min   MVAR(unsigned char,0x300002)
#define hour  MVAR(unsigned char,0x300004)
#define day   MVAR(unsigned char,0x300006)
#define mon   MVAR(unsigned char,0x300008)
#define year  MVAR(unsigned char,0x30000A)
#define week  MVAR(unsigned char,0x30000C)

struct uhr uhr;                // Uhrzeit
struct datum datum;            // Datum
extern struct eingabe eingabe;  // Tastaturpuffer

// Berechnung der Schaltjahre aus dem Jahr
int schaltjahr(BYTE jahr)
{
   return (((1900+(int)jahr % 4 == 0) && (1900+(int)jahr % 100 !=100))
          ||(1900+(int)jahr % 400 == 0));
}


// Berechnung der Tagesnummer aus dem Jahr,Monat und Tag
int tagesnummer(BYTE jahr, BYTE monat, BYTE tag)
{
   int dh,eh;
   dh=((int)monat+10)/13;
   eh=(int)tag+(611*((int)monat+2))/20-2*dh-91;
   return eh+schaltjahr(jahr)*dh;
}


// Berechnung der Wochentage (0=So, 1=Mo,... 6=Sa)
// aus dem Jahr und der Tagesnummer
BYTE wochentag_im_jahr(BYTE jahr, int tg)
{
  int jh,ch;
  jh = ((int)jahr+1900-1)%100;
  ch = ((int)jahr+1900-1)/100;
  return (BYTE)((28+jh+tg+(jh/4)+(ch/4)+5*ch) % 7);
}


// Ermittelt die Anzahl der Tage eines Monats
BYTE tage_von_monat(BYTE monat)
{
 switch(monat)
 {
 case (1):return 31;
 case (2):return 28;
 case (3):return 31;
 case (4):return 30;
 case (5):return 31;
 case (6):return 30;
 case (7):return 31;
 case (8):return 31;
 case (9):return 30;
 case (10):return 31;
 case (11):return 30;
 case (12):return 31;
 }
 return falsch;
}
```

```
// Gibt das letzte im Tastatur-Buffer befindliche Zeichen
// am Bildschirm aus (im Uhrzeit-Format) und positioniert
// den Cursor nach.
void Zeit_Eingabe_anzeigen(void)
{
 BYTE x=9,y=3;

 if(eingabe.pos>6) {y=4;x=x-9;} // Nach dem 6. Zeichen (jahr mon tag)
                                // muß in die 2. Zeile gesprungen werden.
 x=(BYTE)((int)x+(int)eingabe.pos+((int)eingabe.pos-1)/2);
                                // x-Position berechnen (nach jedem
 gotoxy(x,y);                   //  2. Zeichen ein Lehrzeichen)
 write_char(eingabe.puffer[eingabe.pos-1]);  // .. Ausgabe

 x=9;y=3;                          // gleiche Berechnung für ...
 if(eingabe.pos+1>6) {y=4;x=x-9;} // ... die Cursorpositionierung (pos+1)
 x=(BYTE)((int)x+(int)eingabe.pos+1+((int)eingabe.pos)/2);
 setze_position(x,y);
}

// Lädt das Datum in den Timerbaustein
int datum_setzen(struct datum dat)
{
 BYTE tage;
 tage=tage_von_monat(dat.monat);
 if(dat.tag<=tage)          // Struktur auf Richtigkeit überprüfen
      {
       Zehn=0;              // Einerstellen speichern
       day=dat.tag-(dat.tag/10)*10;
       mon=dat.monat-(dat.monat/10)*10;
       year=dat.jahr-(dat.jahr/10)*10;
       week=wochentag_im_jahr(dat.jahr,tagesnummer(dat.jahr,dat.monat,dat.tag));
       Warte();

       Zehn=1;              // Zehnerstellen speichern
       day=dat.tag/10;
       mon=dat.monat/10;
       year=dat.jahr/10;
       Warte();
       return richtig;
      }
 return falsch;
}

// lädt die Zeit in den Timerbaustein
int zeit_setzen(struct uhr zeit)
{
 if(zeit.sekunde<60)
     if(zeit.minute<60)  // Struktur auf Richtigkeit überprüfen
        if(zeit.stunde<24)
         {
          Zehn=0;        // Einerstellen speichern
          sec=zeit.sekunde-(zeit.sekunde/10)*10;
          min=zeit.minute-(zeit.minute/10)*10;
          hour=zeit.stunde-(zeit.stunde/10)*10;
          Warte();
          Zehn=1;        // Zehnerstellen speichern
          hour=zeit.stunde/10;
          sec=zeit.sekunde/10;
          min=zeit.minute/10;
          Warte();
          return richtig;
         }
 return falsch;
}

// Tastaturpuffer auslesen und in Zeit - Struktur speichern
void eingabe_in_zeit(struct uhr *zeit)
{
 zeit->stunde=(eingabe.puffer[6]-'0')*10+eingabe.puffer[7]-'0';
 zeit->minute=(eingabe.puffer[8]-'0')*10+eingabe.puffer[9]-'0';
 zeit->sekunde=(eingabe.puffer[10]-'0')*10+eingabe.puffer[11]-'0';
}

// Tastaturpuffer auslesen und in Datum - Struktur speichern
void eingabe_in_datum(struct datum *dat)
{
 dat->tag=(eingabe.puffer[0]-(BYTE)'0')*10+eingabe.puffer[1]-(BYTE)'0';
 dat->monat=(eingabe.puffer[2]-(BYTE)'0')*10+eingabe.puffer[3]-(BYTE)'0';
 dat->jahr=(eingabe.puffer[4]-(BYTE)'0')*10+eingabe.puffer[5]-(BYTE)'0';
}
```

```
// zwei 4 Bit Werte des Timerbausteines in eine Zahl umwandeln
BYTE lies_Ziffer(BYTE einer,BYTE zehner)
{
 BYTE dif;
 dif=einer/16;           // Die oberen 4 Bit der Einerstelle abschneiden
 einer=einer-dif*16;     //    (enthalten undefinierte Werte)
 dif=zehner/16;          // Zehnerstelle
 zehner=zehner-dif*16;
 return zehner*10+einer; // gesamte Zahl zurückgeben
}

// Eingabe von Datum und Uhrzeit
// Eingabe noch nicht beendet    -> undefiniert
// -- " -- falsch                -> falsch
// -- " -- korrekt               -> richtig
DatumZeit_eingeben(void)
{
 struct uhr zeit;
 struct datum dat;
 if (eingabe.pos==12)        // Ist Eingabe komplett..
    {
     eingabe_in_datum(&dat); // .. Tastaturpuffer in Datum
     eingabe_in_zeit(&zeit); // und Uhrzeit umrechnen
     eingabe.pos=0;          // und löschen
     if (zeit_setzen(zeit))  // Zeit/Datum in Timerbaustein..
       if (datum_setzen(dat))    // .. laden und ..
              {              // .. anzeigen.
              zeit_anzeigen();
              datum_anzeigen();
              return richtig;
              }
     setze_position(10,3);     // Bei Fehleingabe Cursor wieder auf
     return falsch;            // auf Anfangsposition setzen
    }
 else Zeit_Eingabe_anzeigen(); // Bei unvollständiger Eingabe diese
 return undefiniert;           // anzeigen und undefiniert zurückgeben
}

// Aktuelle Zeit ausgeben
void zeit_anzeigen()
{
 BYTE zeit[9];
 zeit[6]=uhr.sekunde/10+'0';  // Aktuelle Zeit in String umwandeln
 zeit[7]=uhr.sekunde -(uhr.sekunde/10)*10+'0';
 zeit[5]=(BYTE)':';
 zeit[3]=uhr.minute/10+'0';
 zeit[4]=uhr.minute -(uhr.minute/10)*10+'0';
 zeit[2]=(BYTE)':';
 zeit[0]=uhr.stunde/10+'0';
 zeit[1]=uhr.stunde -(uhr.stunde/10)*10+'0';
 zeit[8]=0;
 if (sperren())
    {
     printxy(12,0,zeit);     // und, falls Anzeige derzeit nicht
     freigeben();            // benutzt, ausgeben.
     position();             // ursprüngliche Cursorposition wieder
    }                        // herstellen
}

// Datum ausgeben
void datum_anzeigen()
{
 BYTE dat[11];
 dat[0]=(BYTE)'1';
 dat[1]=(BYTE)'9';
 dat[2]=datum.jahr/10+'0';     // Datum in String umwandeln
 dat[3]=datum.jahr -(datum.jahr/10)*10+'0';
 dat[4]=(BYTE)'-';
 dat[5]=datum.monat/10+'0';
 dat[6]=datum.monat -(datum.monat/10)*10+'0';
 dat[7]=(BYTE)'-';
 dat[8]=datum.tag/10+'0';
 dat[9]=datum.tag -(datum.tag/10)*10+'0';
 dat[10]=0;
 if(sperren())
     {
     printxy(0,0,dat);
     freigeben();             // und ausgeben (wie zeit_anzeigen)
     position();
     }
}
```

```
// Zeit & Datum vom Timerbaustein abfragen und bei
// einer Änderung anzeigen
void uhr_abfragen(void)
{
 BYTE sekunde,tag;
 BYTE einer,zehner;

 sekunde=uhr.sekunde;
 tag=datum.tag;

Warte();       // Sekunden lesen
 Zehn=0;
 einer=sec;    // Einerstelle
Warte();
 Zehn=1;
 zehner=sec;   // Zehnerstelle
 uhr.sekunde= lies_Ziffer(einer,zehner); // zusammensetzen und zuweisen

 if (!(sekunde==uhr.sekunde))  // Hat sich die Sekunde geändert...
   {
       Warte();              // Minute lesen
        Zehn=0;
        einer=min;
       Warte();
        Zehn=1;
        zehner=min;
        uhr.minute= lies_Ziffer(einer,zehner);

       Warte();              // Stunde lesen
        Zehn=0;
        einer=hour;
       Warte();
        Zehn=1;
        zehner=hour;
        uhr.stunde= lies_Ziffer(einer,zehner);

       Warte();              // Tag lesen
        Zehn=0;
        einer=day;
       Warte();
        Zehn=1;
        zehner=day;
        datum.tag= lies_Ziffer(einer,zehner);

        zeit_anzeigen();       // Zeit anzeigen
       printf ("Zeit: %i %i  %i ",(int)uhr.stunde,(int)uhr.minute,(int)uhr.sekunde);
       printf ("    Datum: %i %i 19%i     ",(int)datum.tag,(int)datum.monat,(int)datum.jahr);
       switch(datum.woche)    // Zeit/Datum am Terminal anzeigen
          {
          case 1:printf("  Montag");break;
          case 2:printf("  Dienstag");break;
          case 3:printf("  Mittwoch");break;
          case 4:printf("  Donnerstag");break;
          case 5:printf("  Freitag");break;
          case 6:printf("  Samstag");break;
          case 0:printf("  Sonntag");break;
          }
      printf("\n");
      if (!(tag==datum.tag)) // Hat sich der Tag geändert...
            {
             Warte();       // Monat lesen
              Zehn=0;
              einer=mon;
             Warte();
              Zehn=1;
              zehner=mon;
              datum.monat= lies_Ziffer(einer,zehner);

             Warte();        // Jahr lesen
              Zehn=0;
              einer=year;
             Warte();
              Zehn=1;
              zehner=year;
              datum.jahr= lies_Ziffer(einer,zehner);

             Warte();
              Zehn=0;        // Wochentag lesen
              datum.woche= lies_Ziffer(week,0);
             Warte();
```

```
                datum_anzeigen(); // Datum anzeigen
            }
    }
}


 // Überprüft ob sich die aktuelle Zeit innerhalt der vorgegebenen
 // Grenzen befindet
int zeit_vergleichen(struct uhr begin,struct uhr ende)
{
 double zeit,beg,end;
 zeit=(double)uhr.sekunde+60*((double)uhr.minute+60*(double)uhr.stunde);
 beg= (double)begin.sekunde+60*((double)begin.minute+60*(double)begin.stunde);
 end= (double)ende.sekunde+60*((double)ende.minute+60*(double)ende.stunde);
       // in absolute Werte umrechnen und vergleichen..
 if ((beg<=zeit)&&(zeit<=end)) return richtig;
 if ((beg<=zeit)&&(end<beg)) return richtig;
 if ((zeit<=beg)&&(zeit<=end)&&(end<beg)) return richtig;

 return falsch;
}


// Liefert richtig falls Wochenende erreicht
int wochenende(void)
{
 return ((datum.woche==0)||(datum.woche==6));
}
```

```
/*
    DEICHSTETTER Harald                              MAT.:9555258

    Mikrocontroller-Steuerung : ALARMANLAGE

    Time.h                                                    */


void Zeit_Eingabe_anzeigen(void);
 // Gibt das letzte im Tastatur-Buffer befindliche Zeichen
 // am Bildschirm aus (im Uhrzeit-Format) und positioniert
 // den Cursor nach
int DatumZeit_eingeben(void);
 // Eingabe von Datum und Uhrzeit
 // Eingabe noch nicht beendet    -> undefiniert
 // -- " -- falsch                -> falsch
 // -- " -- korrekt               -> richtig
void eingabe_in_datum(struct datum *dat);
 // Tastaturpuffer auslesen und in Datum - Struktur speichern
void eingabe_in_zeit(struct uhr *dat);
 // Tastaturpuffer auslesen und in Zeit - Struktur speichern
void zeit_anzeigen();
 // Aktuelle Zeit ausgeben
void datum_anzeigen();
 // Datum ausgeben
int wochenende();
 // Liefert richtig falls Wochenende erreicht
void uhr_abfragen(void);
 // Zeit & Datum von Timerbaustein abfragen und bei
 // Änderung anzeigen
int zeit_vergleichen(struct uhr begin,struct uhr ende);
 // Überprüft ob sich die aktuelle Zeit innerhalt der vorgegebenen
 // Grenzen befindet
```

Tastatur - Modul

```
/*
    DEICHSTETTER Harald                          MAT.:9555258

    Mikrocontroller-Steuerung : ALARMANLAGE

    Tastatur - Modul
    ================
    Einlesen von Zeichen,Codes, und Zahlen von der Tastatur
                                                              */

#include <string.h>
#include "def.h"
#include "main.h"
#include "ausgabe.h"

#define ungueltig 99

struct eingabe eingabe;     // Tastaturpuffer
extern int zustand;         // Controllerzustand
extern int admin;           // Anzeige ob Administrator angemeldet

BYTE ziffer=0,              // gedrückte Taste
     cnt=0,                 // Hilfsvariablen zum ...
     schritt=0;             // ... Einlesen der Tasten

//  Liefert gedrückte Taste
//  falls keine Taste gedrückt wurde -> undefiniert
//  Der Tastaturpuffer wird wieder gelöscht
int taste(void)
{
 if(eingabe.pos==0) return undefiniert;  // falls Tastaturpuffer leer -> undefiniert
 eingabe.pos=0;                         // Tastaturpuffer wieder löschen
 return eingabe.puffer[0]-(BYTE)'0';     // Taste zurückgeben(als Zahl)
}


// Liest zweistellige Ziffer an Position x,y mit
// maximaler Größe max ein.
// Wurde noch nicht fertig eingelesen -> undefiniert
// Eingelesene Zahl > max             -> ausserhalb
int eingabe_in_nummer(BYTE x,BYTE y,BYTE max)
{
 BYTE nr;
                                        // Letztes im Tastaturpuffer
 gotoxy(x+eingabe.pos,y);                // befindliches Zeichen
 write_char(eingabe.puffer[eingabe.pos-1]); // ausgeben.
 setze_position(x+eingabe.pos+1,y);     // Cursor nachpositionieren
 if(eingabe.pos==2)                     // Wurde 2. Stelle erreicht ..
    {                                   // Tastaturpuffer in Zahl umwandeln
     nr=(eingabe.puffer[0]-(BYTE)'0')*10+eingabe.puffer[1]-(BYTE)'0';
     taste();                           // und löschen.
     if (nr<=max) return nr;            //  falls <= max ... Zahl zurückgeben
     return ausserhalb;                 //  ansonsten ausserhalb
    }
 return undefiniert;                    // falls zweistellige Ziffer noch nicht komplett
}                                       //    -> undefiniert


// Code an Position x,y einlesen  (max 6 Zeichen lange)
// und zurückgeben (wird mit ENTER abgeschlossen)
// Wurde noch nicht fertig eingelesen -> undefiniert
long code_eingabe(BYTE x,BYTE y)
{
 char sterne[7]="*******";
 long nr=0,factor=1;
 int i;
 if((eingabe.puffer[eingabe.pos-1]==(BYTE)'*'))
    {                                   // Falls letztes eingelesenes Zeichen = ENTER
      for(i=eingabe.pos-2;i>-1;i--)
            {                           // Tastaturpuffer in Zahl umwandeln (HEX -> DEC)
             if ((eingabe.puffer[i]>='A') && (eingabe.puffer[i]<='D'))
                         eingabe.puffer[i]=eingabe.puffer[i]-(BYTE)'A'+(BYTE)'9'+1;
                                        // A=10   B=11   C=12 ...
             nr=nr+(long)(eingabe.puffer[i]-(BYTE)'0')*factor;
            factor=factor*16;
            }
      setze_position(x,y+2);            // Cursorposition auf nächste Zeile setzen ...
      taste();                          // ... Tastaturpuffer löschen ...
```

```
      return nr;                          // ... Zahl zurückgeben
    }
 else                                     // Anderenfalls ...
   {
    if ((eingabe.pos==7)) eingabe.pos=6;    // Eingabe darf max 6 Zeichen lang werden
    else
       {
       if (((zustand==Codeabfrage)||(zustand==Alarmcode)||(zustand==Codeaendern))
             &&((y==2)||(y==3)))
              {
               sterne[eingabe.pos]='\0';
               printxy(x,y,sterne);
              }
       else {
              printxy(x,y,eingabe.puffer); // Eingabepuffer ausgeben ..
            }
        setze_position(x+eingabe.pos,y+1);  // .. und Cursor nachpositionieren.
       }
   }
 return undefiniert;                      // Falls Eingabe noch nicht komplett  -> undefiniert
}


// Tastatur initialisieren
void tastatur_initialisieren(void)
{
  T4CON = 0x0040;
  // Timer 4 :  Prescaler factor = 16,  Timer-Betrieb  up,  T2R=0,   Relode-Wert=55000
  // (8*2^1)/16Mhz*(65536-55000) =10,536 ms   -> Tastaturtakt= 94,91268 Hz
  strcpy("",eingabe.puffer); // Tastaturpuffer löschen
  eingabe.pos=0;           // Position = 0
}


// Eingelesene Taste in Zeichen umwandeln
// Sonderzeichen (a,b,c,d) werden nur bei bestimmten Controllerzuständen zugelassen
// ESC bewirkt einen Rücksprung ins Hauptmenü bzw. ...
// ... zur Alarm/Statusanzeige (je nach Controllerzustand)
void taste_behandeln(void)
{
 BYTE zeichen=(BYTE)' ';
 if (eingabe.pos<MAX)                      // maximale Länge des Puffers = MAX
    {
     switch(ziffer)
       {
       case (0): zeichen=(BYTE)'1';break;   // Eingelesene Taste in Zeichen umwandeln ..
       case (1): zeichen=(BYTE)'4';break;
       case (2): zeichen=(BYTE)'7';break;
       case (3): zeichen=(BYTE)'X';break;
       case (4): zeichen=(BYTE)'2';break;
       case (5): zeichen=(BYTE)'5';break;
       case (6): zeichen=(BYTE)'8';break;
       case (7): zeichen=(BYTE)'0';break;
       case (8): zeichen=(BYTE)'3';break;
       case (9): zeichen=(BYTE)'6';break;
       case (10): zeichen=(BYTE)'9';break;
       case (11): zeichen=(BYTE)'*';break;
       case (12): zeichen=(BYTE)'A';break;
       case (13): zeichen=(BYTE)'B';break;
       case (14): zeichen=(BYTE)'C';break;
       case (15): zeichen=(BYTE)'D';break;
    }
     eingabe.puffer[eingabe.pos]=zeichen;    // .. und in Tastaturpuffer schreiben
     eingabe.pos++;
     eingabe.puffer[eingabe.pos]='\0';
    }
  if (zeichen==(BYTE)'X')                    // falls Zeichen = ESC
    {
     if( (zustand==Alarm)||(zustand==Alarmcode) ) zustand=Alarm;
     else if( (zustand==Status)||(zustand==Codeabfrage) ) zustand=Status;
     else  zurueck();    // je nach Zustand zurück zu Alarm, Status oder Hauptmenü (Admin/normal)
     taste();
     Menue_anzeigen();                       // Bildschirminformationen neu anzeigen
    }
  if ( ((zeichen==(BYTE)'A')||(zeichen==(BYTE)'B')||(zeichen==(BYTE)'C')||(zeichen==(BYTE)'D'))
     && !((zustand==Codeabfrage)||(zustand==Alarmcode)||(zustand==Alarm)||(zustand==Status)
          ||(zustand==Codeaendern)||(zustand==Klassenauswahl)||(zustand==Codeneu)))
      eingabe.pos--;
 else                          // Die Zeichen A,B,C,D sind nicht immer zulässig -> ausfiltern
    zustand_abfragen();        // Controllerzustand mit eingelesener Taste behandeln
}
```

```
// Simulation der Tastatur für Simulationsumgebung
// void tastatur_interrupt(void) interrupt 0x24 using regbank_003
// {
//  BYTE alt;
//  alt=ziffer;
//  if (TE0) ziffer=7;
//  else if (TE1) ziffer=0;
//  else if (TE2) ziffer=4;
//  else if (TE3) ziffer=8;
//  else if (TE4) ziffer=1;
//  else if (TE5) ziffer=5;
//  else if (TE6) ziffer=9;
//  else if (TE7) ziffer=2;
//  else if (TE8) ziffer=6;
//  else if (TE9) ziffer=10;
//  else if (ENT) ziffer=3;
//  else if (ESC) ziffer=11;
//  else if (A) ziffer=12;
//  else if (B) ziffer=13;
//  else if (C) ziffer=14;
//  else if (D) ziffer=15;
//  else ziffer=ungueltig;
//  if ((ziffer!=alt)&&(ziffer!=ungueltig)) taste_behandeln();
// }


// Liest Zeichen von der Tastatur
void tastatur_interrupt(void) interrupt 0x24 using regbank_003
{
   BYTE alt;
   T4=55000;       // Timer - Relode
   alt=ziffer;

   if (TE) {ziffer=schritt*4;cnt=5;}    // Reihe der Tastatur abfragen
   if (TF) {ziffer=schritt*4+1;cnt=5;}
   if (TD) {ziffer=schritt*4+2;cnt=5;}
   if (TC) {ziffer=schritt*4+3;cnt=5;}

   cnt--;          // Ständiges Drücken wird nur als 1 Zeichen erkannt
   if (cnt<1) ziffer=ungueltig;
   schritt++;
   if (schritt==4) schritt=0;
   switch(schritt)  // Spalten der Tastatur ansteuern
     {
     case (0): TA=richtig;TH=falsch;break;
     case (1): TB=richtig;TA=falsch;break;
     case (2): TG=richtig;TB=falsch;break;
     case (3): TH=richtig;TG=falsch;break;
     }            // Falls sich Taste geändert hat && nicht ungültig ist -> Taste behandeln
   if((alt!=ziffer)&&(ziffer!=ungueltig)) taste_behandeln();

}
```

```
/*  DEICHSTETTER Harald                          MAT.:9555258

    Mikrocontroller-Steuerung : ALARMANLAGE

    Tastatur.h                                                */

int taste(void);
 //  Liefert gedrückte Taste
 //  falls keine Taste gedrückt wurde -> undefiniert
 //  Der Tastaturpuffer wird wieder gelöscht
int eingabe_in_nummer(BYTE x,BYTE y,BYTE max);
 // Liest zweistellige Ziffer an Position x,y mit
 // maximaler Größe max ein.
 // Wurde noch nicht fertig eingelesen -> undefiniert
 // Eingelesene Zahl > max             -> ausserhalb
void tastatur_initialisieren(void);
 // Tastatur initialisieren
long code_eingabe(BYTE x,BYTE y);
 // Code an Position x,y einlesen  (max 6 Zeichen lange)
 // und zurückgeben (wird mit ENTER abgeschlossen)
 // Wurde noch nicht fertig eingelesen -> undefiniert
void tastatur_interrupt(void);
 // Liest Zeichen von der Tastatur
```

Anzeige - Modul (4Bit-Interface)

```
/*
    DEICHSTETTER Harald                              MAT.:9555258

    Mikrocontroller-Steuerung : ALARMANLAGE

    Ausgabe - Modul
    ===============
    Ansteuern eines 4*20 Zeichen LCD- Display über Portleitungen
    des C161                                                                */

#include <stdlib.h>
#include <stdio.h>
#include <reg161.h>
#include <absacc.h>
#include "def.h"
#include "time.h"

int benutzt;                // richtig falls Anzeige gerade benutzt

BYTE posx=0,posy=0;// Gespeicherte aktuelle x und y Position
                           // ( um nach Zeit/Datum-Anzeige wieder
                           //   zur gewünschten Position zurückzugekehren)

// Wartezeit ( wird für Anzeige benötigt)
void Warte(void)
{
   int i;
   for (i=0; i<=1500;i++){};
}


// physikalisches Speicher-Modell initialisieren
void controller_initialisieren (void)
{
  BUSCON0 |= 0x003F;  //0x001D;    /*; 0 Wait no Delay */

  /* Serial interface initialisieren  */
  P3  |= 0x0400;        /* SET PORT 3.10 OUTPUT LATCH (TXD)              */
  DP3 |= 0x0400;        /* SET PORT 3.10 DIRECTION CONTROL (TXD OUTPUT)  */
  DP3 &= 0xF7FF;        /* RESET PORT 3.11 DIRECTION CONTROL (RXD INPUT) */
  S0TIC = 0x80;         /* SET TRANSMIT INTERRUPT FLAG                   */
  S0RIC = 0x00;         /* DELETE RECEIVE INTERRUPT FLAG                 */
  S0BG  = 0x33;         /* SET BAUDRATE TO 9600 BAUD AT 16MHZ            */
  S0CON = 0x8011;       /* SET SERIAL MODE                               */

  /* Bus für LCD - Display initialisieren*/
   BUSCON1 = 0xc6F0;     /*1100 0011 1111 0000
                         II    II  IIII I MCTC=15 waitstates
                         II    II  IIII RWDC read/write delay
                         II    Ii  IIIMTTC 1 waitstate
                         II    II  IIBTYP 16 bit Mult
                         II    II  IALECTL Lengthened ALE
                         II    II  BUSACT externer Bus
                         II    IRDYEN MCTC only
                         IICSREN 1
                         I CSWEN  1         */
  ADDRSEL1 = 0x2000;  // 0x20:0000    4kb -Bereich

 /* Bus fuer Timer initialisieren */
   BUSCON2 = 0xc6F0;     /*1100 0011 1111 0000
                         II    II  IIII I MCTC=15 waitstates
                         II    II  IIII RWDC read/write delay
                         II    Ii  IIIMTTC 1 waitstate
                         II    II  IIBTYP 16 bit Mult
                         II    II  IALECTL Lengthened ALE
                         II    II  BUSACT externer Bus
                         II    IRDYEN MCTC only
                         IICSREN 1
                         I CSWEN  1   */
  ADDRSEL2 = 0x3000;  // 0x30:0000    4kb -Bereich
}
```

```
void Enable(void)
{
 En=0;
 Warte();
 En=1;
 Warte();
 Warte();
}

void out_Port(BYTE wert)
{
 Dat0=!(wert & 1);
 Dat1=!(wert & 2);
 Dat2=!(wert & 4);
 Dat3=!(wert & 8);
 Warte();
}

// Wert ins Steuerregister des LCD-Displays schreiben
void DisplayControll(BYTE wert)
{
 RSelect=1;
 out_Port(wert / 16);
 Enable();
 out_Port(wert);
 Enable();
}

// Wert ins Datenregister des LCD-Displays schreiben
void DisplayDat(BYTE wert)
{
 RSelect=0;
 out_Port(wert / 16);
 Enable();
 out_Port(wert);
 Enable();
}

// Anzeige sperren
// Wird die Anzeige gerade benutzt -> falsch
// anderenfalls                    -> richtig
int sperren(void)
{
 if (benutzt) return falsch; // Falls bereits benutzt  ->falsch
 benutzt=richtig;          // anderenfalls sperren
 return  richtig;          //        -> richtig
}

// Anzeige freigeben
void freigeben(void)
{
 benutzt=falsch;
}

// Zeichen auf Anzeige ausgeben
void write_char (BYTE c)
{
   if (c==(BYTE)'ü') c = (BYTE) 0xf5;       // ASCII Zeichen in Display Zeichen
   else if (c==(BYTE)'ö') c = (BYTE)0xef;   // umwandeln
   else if (c==(BYTE)'ä') c = (BYTE)0xe1;
   DisplayDat((BYTE)c);                         // in Datenregister der Anzeige laden
}

void ausgabe_loeschen(void)
{
   DisplayControll((BYTE)0x01);         /* Display löschen               */
}

// Initialisiert das LCD - Display
void ausgabe_initialisieren(void)
{
  En=1;
  Warte(); Warte(); Warte();
  RSelect=1;
  out_Port(2);
  Enable();
  DisplayControll((BYTE)0x28);         /* 4 -Bit, vierzeilig, 5x7 Punkte */
  ausgabe_loeschen();                  /* Anzeige neu initialisieren */
  DisplayControll((BYTE)0x0f);
}
```

```
// Cursor positionieren
// Ursprung liegt bei 0,1
void gotoxy(BYTE x_col, BYTE y_row)
{
   BYTE wert=0;

   switch (y_row) {
   case 1 : wert = (BYTE)0x00;
              break;
   case 2 : wert = (BYTE)0x40;
              break;
   case 3 : wert = (BYTE)0x14;
              break;
   case 4 : wert = (BYTE)0x54;              // Startadresse der Reihe berechnen
             break;
   }
   wert = (wert + x_col) | (BYTE)0x80;      // Spaltenadresse addieren und zusätzliche
                                            // Bits setzen
   DisplayControll(wert);
}


// Gibt an der Position spalte,zeile den angegebenen String txt aus.
// Ursprung liegt bei 0,0
void printxy(BYTE spalte,BYTE zeile,BYTE *txt)
{
  gotoxy(spalte,zeile+1);
  while (*txt != 0)
            write_char (*(txt++));          // Zeichen bis zum Terminierungssymbol ausgeben.
}


// Cursor einschalten
// Bildschirm wird gelöscht und anschließend Datum+Uhrzeit ausgegeben
void cursor_an(void)
{
   ausgabe_loeschen();                      /* Anzeige neu initialisieren */
   DisplayControll((BYTE)0x0f);             /* Cursor an */
   datum_anzeigen();                        /* Datum und Uhrzeit ausgeben */
   zeit_anzeigen();
   zeit_anzeigen();
}


// Cursor ausschalten
// Bildschirm wird gelöscht und anschließend Datum+Uhrzeit ausgegeben
void cursor_aus(void)
{
   ausgabe_loeschen();                      /* Anzeige neu initialisieren */
   DisplayControll((BYTE)0x0c);             /* Curosor aus*/
   datum_anzeigen();                        /* Datum und Uhrzeit ausgeben */
   zeit_anzeigen();
}


// Zweistellige Nummer an Position x,y ausgeben
void zeige_nummer(BYTE x,BYTE y,BYTE nr)
{
 char txt[3];
 txt[0]=nr/10+'0';
 txt[1]=nr-(nr/10)*10+'0';                // Nummer in Text umwandeln
 txt[2]='\0';
 printxy(x,y,txt);                        // -> ausgeben
}


// Position festlegen
void setze_position(BYTE x,BYTE y)
{
 posx=x;
 posy=y;
 gotoxy(posx,posy);
}


// Position wiederherstellen
// muß nach Anzeige von Datum/Uhrzeit geschehen
void position(void)
{
 gotoxy(posx,posy);
}
```

```
/*
    DEICHSTETTER Harald                          MAT.:9555258

    Mikrocontroller-Steuerung : ALARMANLAGE

    Anzeige.h                                                */


 void ausgabe_initialisieren(void);
  // Initialisiert das LCD - Display

 void gotoxy(BYTE spalte ,BYTE zeile);
  // Cursor positionieren

 void printxy(BYTE spalte,BYTE zeile,char *txt);
  // Gibt an der Position spalte,zeile den angegebenen String txt aus.

 void cursor_an(void);
  // Cursor einschalten
  // Bildschirm wird gelöscht und anschließend das Datum+Uhrzeit ausgegeben

 void cursor_aus(void);
  // Cursor ausschalten
  // Bildschirm wird gelöscht und anschließend das Datum+Uhrzeit ausgegeben

 void zeige_nummer(BYTE x,BYTE y,BYTE nr);
  // Zweistellige Nummer an Position x,y ausgeben

 void write_char (BYTE c);
  // Zeichen auf Anzeige ausgeben

 void setze_position(BYTE x,BYTE y);
  // Position festlegen

 void position(void);
  // Position wiederherstellen
  // muß nach Anzeige von Datum/Uhrzeit geschehen

 int  sperren(void);
  // Anzeige sperren
  // Wird die Anzeige gerade benutzt -> falsch
  // anderenfalls                    -> richtig

 void freigeben(void);
  // Anzeige freigeben

 void controller_initialisieren (void);
  // physikalisches Speicher-Modell initialisieren

 void Warte(void);
  // Wartezeit
```

5. Aufgetretene Probleme bei der Entwicklung der Alarmanlage:

Programmierung:

- Deutliche Unterschiede zwischen den Mikrocontrollertypen 80C161 und 80C167 bezüglich der Programmierung:

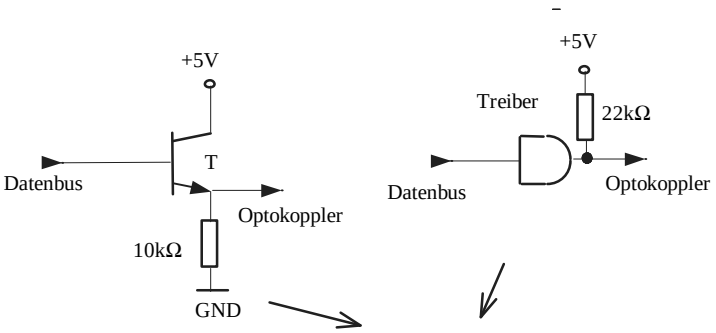
    - Unterschiedliche Registerbelegungen, insbesondere bei der Programmierung des Bus-Timings, bez. der Einstellung der Timer und Interrupt.
      (Das Programm wurde zuerst für den 80C167 Controller erstellt, da hierfür ausreichende Datenbücher vorhanden waren. Später mußte es an den 80C161 angepaßt werden.)

    - Kein automatischer Reload bei Timer-Programmierung bei 80C161
      ➔ komplizierte und ungenaue Timer-Einstellungen
      (im Endeffekt durch Timerbaustein gelöst)
    - Weniger Ports (80C161 : 5 16 Bit und 2 8 Bit Ports
      80C167 : 5 16 Bit und 4 8 Bit Ports)
      ➔ führte zu Engpässen, da ein Großteil der Ports bereits für alternative Funktionen reserviert ist.

- Die Programmierung der Anzeige gestaltete sich als sehr schwierig, da die mir zur Verfügung gestellten Unterlagen einem anderen Typen entsprachen, und ich das lange Zeit nicht wußte.

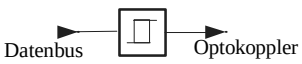
Hardware:

- Die galvanische Entkopplung von Mikrocontroller und den externen Komponenten wie Tastatur, Türkontakte und vor allen der Anzeige führte zu Problemen :
    - Transistoren bzw. Treiber zum Ansteuern der Optokoppler (werden benötigt, da der dieser einen Strom von 10mA aufnimmt) führten zu einen Absturz des Mikrocontrollers. Ein Grund dafür konnte nicht gefunden werden.



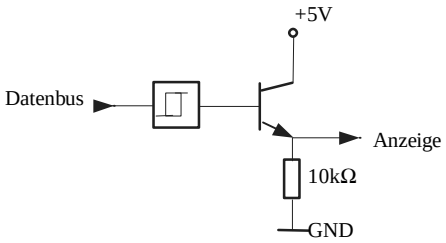
Absturz!

➔ Verwendung eines Busbtreibers (mit Schmitt-Triggereingang)



- Die Verwendung von Optokoppler zur galvanischen Trennung von Anzeige und Controller führte zu keinem zufriedenstellenden Ergebnis. Der Grund könnte die für die Optokoppler möglicherweise zu schnelle Übertragung sein

➔ Entkopplung der Anzeige nur mittels Bustreiber und nachfolgendem Transistor.



- Standard Bus-Timing nicht kompatibel zur Anzeige
  ➔ Definieren eines speziellen Timings innerhalb gewisser Adressbereiche:

```
BUSCON1 = 0xc6c0;
```

1	1	0	0	0	1	1	0	1	1	1	1	0	0	0	0
CSW EN1	CSR EN1				BUS ACT1	ALE CTL1		BTYP		MTT C1	RWD C1	MCTC			

.. 15 waitstates, read/write delay, 8 bit Multixpex, lenghened ALE
(siehe Programm - Berechnung)

```
ADDRSEL1 = 0x2000;    0x20:0000    4kb -Bereich
```

- Die zu langen Datenleitungen zwischen Mikrocontroller und Anzeige/Tastertur führten zu Störungen. Mittels Pegelanpassung, langsamerer Datenübertragung und schließlich noch einem RC-Tiefpaß - Filter konnte eine fehlerfreie Übertragung erreicht werden.

- Der Prototyp - mäßige Aufbau stellte sich immer wieder als großes Problem heraus. Oft lösten sich Leitungen (kalte Lötstellen ...) und eine langwierige Fehlersuche war die Folge.

Allgemein:

Häufig war unklar ob die Ursache eines Fehlers an der Hard- oder Software lag. Das Testen gestaltete sich deshalb als sehr schwierig und auch zeitaufwendig.

6. Fehlerbeseitigung

Aufgetretener Fehler	**Beseitigung**
Mikrocontroller bleibt im Terminal- Mode	• Versorgung prüfen (+5V) • Anschlüsse zu ext. Komponenten prüfen • Bootstrap Mode / normal Program Schalter (Siehe Seite 16) • RESET - Taste drücken • Serielle Verbindung richtig anstecken (Siehe Seite 16)
keine Anzeige sichtbar	• Versorgung prüfen • Anschlüsse prüfen • Mult-/Demultiplex - Schalter (Siehe Seite 16)

7. Stückliste:

Teil	Stück	Benennung
LP1	1	Epoxidharzplatte 160,0*100,0*1,5 35µ Cu einseitig
LP2	2	Epoxidharzplatte 100,0*56,0*1,5 35µ Cu einseitig
C_1-C_6	1	Folienkondensator 33nF
R_{40}-R_{45}	1	Kohleschichtwiderstand 1,5kΩ 1/4 W ±10%
R_{39}	1	Kohleschichtwiderstand 4,7kΩ 1/4 W ±10%
R_{33}-R_{38}	6	Kohleschichtwiderstand 470Ω 1/4 W ±10%
R_{17}-R_{32}	16	Kohleschichtwiderstand 10kΩ 1/4 W ±10%
R_1-R_{16}	16	Kohleschichtwiderstand 270Ω 1/4 W ±10%
P_1	1	Potentiometer 10kΩ
D_1-D_4	4	Diode 1N4007
T_1-T_{11}	11	npn - Transistor BD137
IC_{28}	1	Timer RTC 72421A
IC_{27}	1	Inverter 74LS05
IC_{25}-IC_{26}	1	Treiber GD 74LS07
IC_{21}-IC_{24}	4	JK-Master-Slave-FF 74LS109
IC_{29},IC_{30}		
IC_{19},IC_{20}	4	Bustreiber SN74LS245 (TI)
IC_9,IC_{10}	2	ODER-Gatter (4 Eingänge) MC14072BCP (8274)
IC_{11}-IC_{18}		
IC_1-IC_8	16	Optokoppler PC817 (SHARP)
	4	IC-Sockel 20-polig Reihenabstand 7,62
	1	IC-Sockel 18-polig Reihenabstand 7,62
	6	IC-Sockel 16-polig Reihenabstand 7,62
	7	IC-Sockel 14-polig Reihenabstand 7,62
	8	IC-Sockel 4-polig Reihenabstand 7,62
Anzeige	1	LCD-Anzeige DV-20400 20 Zeichen * 4 Zeilen
Tastatur	1	Tastenblock 16 Tasten
	1	Stiftleiste 80-polig gerade
	1	Buchsenleiste 80-polig gerade
	8	Printklemmen 2-polig Pinabstand 5,08
	8	Reed - Relais
	1	Piezo -Summer
	1	Flachbandkabel 1m Rasterabstand 1,27 20 polig
	1	Mikrocontrollerplatine PHYTEC KitCON-161