Christian Beck

DIPLOMARBEIT

Interface für Kommunikationssysteme für Schwerstbehinderte basierend auf IR–Sensoren

ausgeführt am Institut für

Allgemeine Elektrotechnik und Elektronik

der Technischen Universität Wien

unter der Anleitung von

aO. Prof. Dr. E. Wintner, Univ. Prof. Dr. H. Thoma und Ing. G. Prulamp

durch

Christian Beck

Gersthoferstraße 43/2–3; A–1180 Wien

21. Januar 2005 .....

Inhaltsverzeichnis

Vorwort

Diese Dokumentation ist in vier Teile gegliedert, um ein einfaches Lesen und schnelles Auffinden der für den Leser relevanten Teile zu ermöglichen, als auch die inhaltlichen Trennungen für den Betrachter auf einen Blick darzustellen.

Der erste Teil zeigt die Grundlagen der Arbeit und das Verwendungsfeld des Gerätes auf. Technische Details wurden mit Absicht vermieden, sodaß auch weniger technisch Informierte, den Sinn und Zweck sowie die Funktionen der Diplomarbeit verstehen können.

Der zweite Teil zeigt alle theoretischen Überlegungen und Planungen inklusive des Programms und dessen Entwicklung sowie die Hardwareplanungen. Während der ganzen Entwicklung wurde auf die zukunftsorientierte Erweiterbarkeit Wert gelegt, um Verbesserungen und Neuversionen so einfach wie möglich zu machen.

Der dritte Teil zeigt die praktischen Aufbauten, die Fehlersuche und Beseitigung und die ersten Testberichte.

Der vierte und letzte Teil enthält eine Funktionsbeschreibung und eine Bedienungsanleitung für das Gerät, wobei die Bedienungsanleitung auch komplett getrennt von der schriftlichen Diplomarbeit verwendet werden kann.

Formatumsetzung

Diese Diplomarbeit wurde mit LaTeX geschrieben. Da sich das DVI–Format leider nicht durchgesetzt hat, habe ich die Diplomarbeit ins PDF-Format portiert, damit jeder diese Arbeit lesen kann. Verwendet habe ich dazu das kostenlose Programm pdfTeX, daß (wie man sieht) sehr gute Ergebnisse liefert obwohl es noch eine Betaversion ist; bemerkenswert ist weiters, daß alle 203 Seiten mit allen Bildern in Druckqualität nur ca. 3MB benötigen; da kann sich Microsoft ein Beispiel nehmen.

Ich hoffe, daß bei der Portierung keine Fehler aufgetreten sind, aber ich habe leider nicht die Zeit, mir das gesamte Dokument genau durchzusehen; Tippfehler gibt es leider sicher immer noch.

History

Im Zuge der Updates von pdfTEX versuche ich dieses Dokument weiter zu verbessern, obwohl die mir dafür zur Verfügung stehende Zeit nur sehr gering ist. In diesem Abschnitt möchte ich die Verbesserungen[1] auflisten:

Juni 98:

- Neue Übersetzung mit pdfTEX 0.12l
- setzen der Dokumentinfos
- Seitenrandeinstellungen verbessert

Juli 98:

- Tippfehlerverbesserungen
- PS–Font verändert, damit Adobe–Viewer keine Fehler bei Kreisen melden

Jänner 99:

- upgrade auf pdfTEX 0.12r
- Umstellung auf LaTEX 2_ε
- Verwendung neuer type1 fonts

Februar 99:

- Durch Verwendung des „hyperref“–packages volle Unterstützung von Links (Inhaltsverzeichnis, Querverweise, Internetadressen, ...) im PDF–Dokument.

Jänner 05:

- Das *alte* Dokument wurde nochmals auf den neuesten pdfTEX Stand gebracht und somit auch auf einen aktuellen PDF-Stand.

[1] Da es sich um meine Diplomarbeit handelt, wird der Inhalt selbverständlich *nicht* verändert; Verbesserungen betreffen nur die Konversion, Tippfehler u.ä.

Teil I

Allgemeines

Kapitel 1

Einleitung

Dieses Kapitel soll den Leser in die Problematik der technischen Bedürfnisse von Schwerstbehinderten einführen und die Notwendigkeit bzw. Neuerung der Diplomarbeit näher erläutern.

1.1 Schwerstbehinderte

Schwerstbehinderte Personen sind in ihrer körperlichen Freiheit so stark eingeschränkt, daß ihnen das Hantieren mit Standardgeräten nicht möglich ist und sie für tagtägliche „Handgriffe“ wie das Öffnen von Türen, das Bedienen von Unterhaltungselektronikgeräten (Fernsehen, HiFi, Computer), oder auch Dingen wie das Fenster und das Licht spezielle technische Unterstützung benötigen. Beispiele für solche Erkrankungen sind Querschnittslähmung, Multiple Sklerose, Parkinson, Hirn– und Herzschlagfolgen.

1.2 bereits käufliche Geräte

Da die Anzahl der Schwerstbehinderten zum Glück relativ klein ist, ist die Entwicklung spezieller Geräte wirtschaftlich schwer möglich, sodaß es derzeit nur wenige, sehr allgemeine Geräte gibt, die man in drei Kategorien einteilen kann:

- Umweltkontrollsysteme
- Kommunikationssysteme
- Computereingabehilfen

1.2.1 Umweltkontrollsysteme

In diese Kategorie fallen alle Geräte, die dem Patienten eine Veränderung seiner Umwelt ermöglichen; darunter ist auch die Bedienung von Geräten jeder Art zu verstehen. Am Markt befinden sich derzeit vorwiegend spezielle lernfähige Infrarotfernsteuerungen (wie in Abbildung 1.1 dargestellt), und infrarotfernsteuerbare Geräte wie Bettverstellungen,

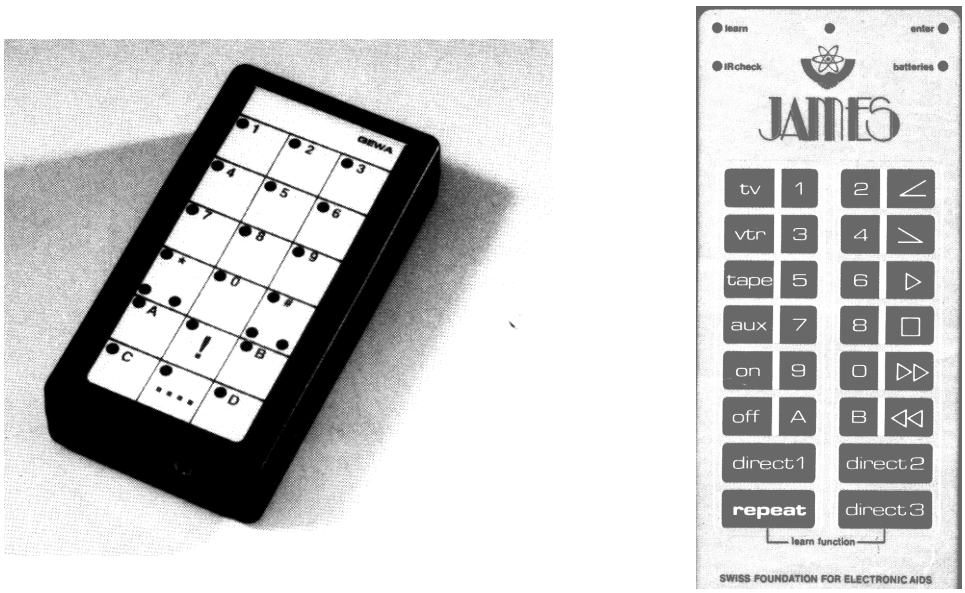

Abbildung 1.1: Beispiele für lernfähige Infrarotfernsteuerungen: links GEWA Prog [5] — rechts James [6]

Rolläden, Türöffner, Telefone udgl. Für nicht besonders adaptierte Geräte kann eine einfache Ein/Aus–Funktion mit Hilfe infrarotgesteuerter Steckdosen (siehe Abbildung 1.2) erreicht werden; Licht wird ebenso mit entsprechenden Schaltern (siehe Abbildung 1.2)

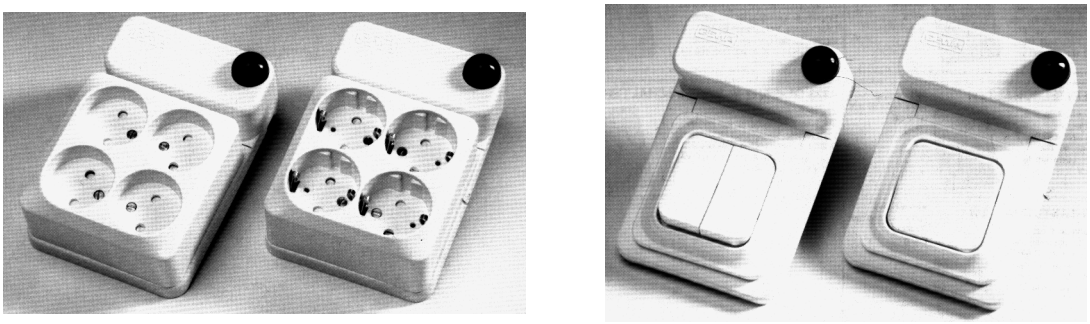

Abbildung 1.2: Beispiele für infrarotfernsteuerbare Elektroinstallationen [5]

ferngesteuert bedient.

Weiters zählen zu den Umweltkontrollsystemen die *Alarm– und Notrufsysteme*. Schwerstbehinderte sind trotz ihrer Erkrankung mit Hilfe von elektrischen Rollstühlen in der Lage, sich fortzubewegen und dabei zum Beispiel in einem Parkgelände eigenständig umherzufahren. Dabei kann es zu Situationen kommen, die einen Alarm– oder Notruf nötig machen.

Alarmrufe treten dann ein, wenn der Patient nicht mehr aus eigener Kraft weiterkommt und fremde Hilfe rufen muß; dies kann zum Beispiel durch ein Steckenbleiben des Rollstuhls passieren.

Notrufe sind solche, die wegen einer lebensbedrohlichen Situation dringend das Handeln von anderen Personen verlangen; die Gründe hierfür können zum Beispiel Atembeschwerden oder Kreislaufprobleme sein. Wichtig ist dabei die sichere einfache Bedienung, da durch eine bedrohliche Situation der Schwerstbehinderte leicht in Panik geraten kann und komplizierte oder schwierige Mechanismen nicht mehr bedienen kann.

1.2.2 Kommunikationssysteme

Durch die Erkrankung kann zusätzlich auch die Sprache des Menschen mehr oder weniger in Mitleidenschaft gezogen sein. Daher benötigen diese Patienten technische Hilfsmittel, mit denen sie mit ihrer Umwelt sprachlich „kommunizieren“ können. Darunter fallen digitale Sprachaufzeichnungsgeräte (Speicherung natürlicher Sprache) und Sprachsysteme, die Text sprechen können (synthetische Sprache). Erstere können besonders häufig benutzte Phrasen speichern und bei Bedarf wiedergeben, zweitere übersetzen jeden Text in Sprache. Ein Beispiel für eine Bedienungsoberfläche für ein Kommunikationsmittel ist in Abbildung 1.3 links zu sehen. Hier erfolgt die Kommunikation durch Auswählen einer Taste auf der

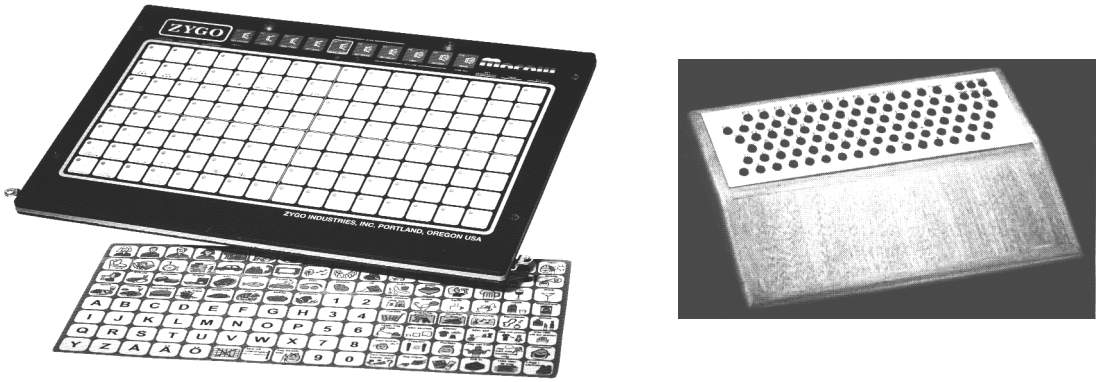

Abbildung 1.3: links: Beispiel für ein digitales Sprachaufzeichnungsgerät mit Großtastatur zur Bedienung [5] — rechts: Beispiel für eine Mikrotastatur [8]

Großtastatur; die Texte werden bei diesem Gerät vorher digital aufgezeichnet.

1.2.3 Computereingabehilfen

Eine in letzter Zeit immer wichtiger werdende Möglichkeit Dinge zu erledigen bzw. Informationen mit seiner Umwelt auszutauschen ist der Computer. Daher gibt es einige Geräte, die dem erkrankten Menschen die Bedienung von Computern ermöglichen. Auf diesem Gebiet gibt es vorwiegend Ersätze für die bekannten Eingabesysteme Mouse und Tastatur.

1.2.3.1 Tastaturen

Die Hilfsmittel zum Ersatz oder zur Verbesserung von Tastaturen sind je nach Schweregrad der Behinderung stark verschieden.

Im einfachsten Fall helfen Tastaturen mit besonders kleinen Abständen (siehe Abbildung 1.3 rechts) zwischen den Tasten, um mit einem sehr kleinen Greifraum alle Standardtasten bedienen zu können. Genauso können Tastaturen mit besonders großen Tasten (siehe Abbildung 1.3 links) Hilfe bei sehr grober Motorik bringen.

Bei schwerwiegenderen Fällen können Sondertastaturen mit einer eingeschränkten Anzahl von Tasten eingesetzt werden, die ohne Handbewegung nur mit Fingerbewegungen bedient werden können; Einzeltasten werden damit durch Tastenkombinationen simuliert.

Zusätzlich können bei allen Arten von Tastaturen Abdeckplatten (siehe Abbildung 1.4) verwendet werden, um ein versehentliches Drücken von Tasten zu vermeiden oder die

Abbildung 1.4: Standardtastatur mit Abdeckung zur Verhinderung von Fehleingaben [8]

Bedienung der Tasten mit Hilfe von Mundstücken zu erleichtern.

Zuletzt gibt es noch die Kategorie der Tastaturemulatoren mit Scanbetrieb[1]. Diese Systeme können sowohl hardware– als auch softwaremäßig realisiert werden.

[1] Für eine prinzipielle Erklärung des Scanbetriebs siehe 1.3.2 auf Seite 12

1.2.3.2 Mouse–Emulatoren

Ein einfacher Ersatz für eine Standard–Mouse ist der Trackball. Der Vorteil gegenüber einer Mouse besteht darin, daß nur die Finger bewegt werden müssen, die Hand aber ruhig liegen bleiben kann; der Trackball erfordert jedoch eine relativ feine Motorik.

Ein weiterer Ersatz besteht in der Verwendung von Touchscreens, wobei direkt mit einer Extremität durch Berührung des Bildschirms die Mouse gesteuert werden kann; die Mouse–Tasten werden entweder softwaremäßig emuliert oder hardwaremäßig durch drei beliebig ansteckbare Sensoren ersetzt.

Die letzte Form des Mouse–Ersatzes ermöglicht die Steuerung der Bewegung durch Einzelsensoren, die auch in einer Art Joystick vereint sein können (siehe Abbildung 1.5); die Bewegungsgeschwindigkeit ist einstellbar.

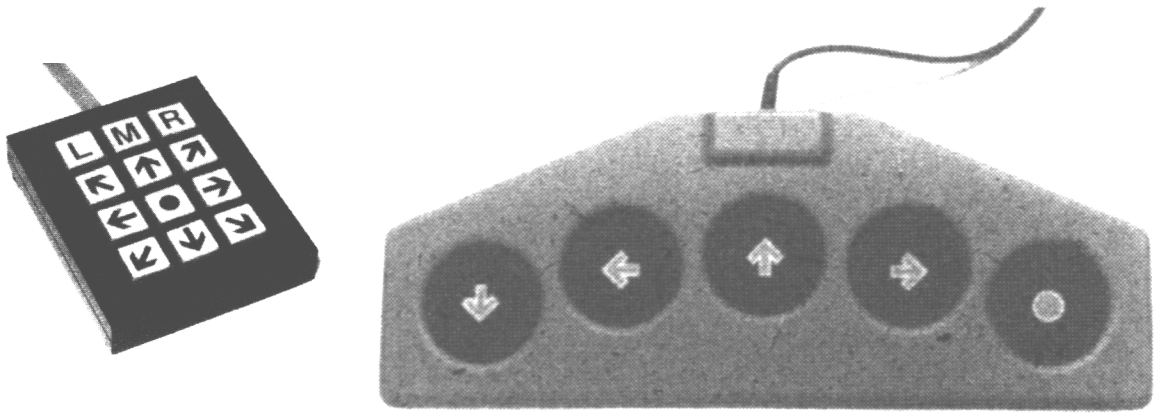

Abbildung 1.5: Beispiele für Mousesysteme mit Einzelsensoren [8]

1.3 Möglichkeiten der Bedienung von Geräten durch Schwerstbehinderte

Schwerstbehinderten bleiben durch ihre starken körperlichen Einschränkungen nur eine der folgenden Bewegungen um Informationen an ihre Umwelt zu übertragen:

- Kopfbewegungen
- Augenbewegungen
- Bewegungen mit dem Mund (Zunge, Blasen, Geräusche, ...)
- sehr grobe Bewegungen mit den Extremitäten

Daher hat man Möglichkeiten entwickelt, alle oben genannten Geräte nur mit diesen Bewegungen zu steuern.

An dieser Stelle möchte ich auch die Problematik der noch nicht erwähnten Sprachsteuerung anschneiden. Derzeit ist es — entgegen anderer Behauptungen mancher Computerfirmen — noch nicht möglich, Computer mit jeder beliebigen Stimme zu steuern. Bestehende Systeme basieren auf einem Trainingssystem, das bestimmte Spracheigenheiten des Benutzers registriert und daraus die gesprochenen Worte ermittelt. Wichtig ist daher für die korrekte Funktion des Entschlüsselungssystems, daß die Sprache des Benützers immer möglichst gleich ist. Diese Voraussetzung ist aber leider bei einem Großteil der Schwerstbehinderten nicht erfüllt. Durch ein Fortschreiten der Krankheit verändert sich die Stimme laufend, oder häufige Erkältungen, durch das geschwächte Immunsystem hervorgerufen, verändern die Stimme. Auch sind solche Systeme völlig unbrauchbar für Notruffunktionen, da sich die Stimme durch die Aufregung stark verändert. Weiters ist die sonst nicht so nachteilige mögliche mehrmalige Wiederholung des Wortes hier sehr gefährlich, da der Patient vielleicht durch den Notfall nicht in der Lage ist, einen Hilferuf mehrmals zu wiederholen.

Ein weiterer Nachteil sprachgesteuerter Systeme besteht in der derzeit immer noch hohen notwendigen Rechnerleistung. Daher können solche Sprachsystem nur in relativ leistungsstarken Computersystemen verwirklicht werden und nicht in kleinen einfach zu bedienenden tragbaren Geräten, wie sie zum Beispiel auf Rollstühlen benötigt werden. Laptops sind zwar vom Prinzip her transportabel, Tests haben aber gezeigt, daß die harten Erschütterungen eines Rollstuhles die mechanische Festigkeit von üblichen Laptops überfordert.

Zuletzt sei noch die prinzipielle Abneigung mancher Patienten gegen Computer erwähnt. Obwohl auch programmierbare Fernsteuerungen vom Aufbau her Kleincomputer sind, ist es ein starker psychologischer Unterschied, ob man ein Gerät, das wie eine Fernsteuerung aussieht, über Tastenkombinationen bedient, oder ob man einen Computer bedienen soll.

1.3.1 Auslösung von Impulsen

Wie ich später noch erläutern werde, kann man eine Steuerung von Geräten durch eine oder mehrere Impulsfolgen verwirklichen. Daher ist es das vorrangige Ziel, möglichst für jede Erkrankungsform eine oder mehrere Möglichkeiten zur Erzeugung von Impulsen zu finden; solche Geräte bzw. Systeme nennt man in diesem Gebiet der Medizintechnik Sensoren.

1.3.1.1 mechanische Sensoren

Mechanische Sensoren sind langläufig als Schalter bzw. Taster bekannt. Diese werden in den verschiedensten Bauformen hergestellt um ein möglichst sicheres, kraftarmes und/oder motorisch einfaches Auslösen zu gewährleisten. Dabei sollen diese Schalter aber auch besondere Robustheit und medizinisch–hygienische Kriterien erfüllen. Als Beispiele seien hier Kissen–, Fuß–, Finger– und Zungenkontakte genannt (siehe Abbildung 1.6 und 1.7 links).

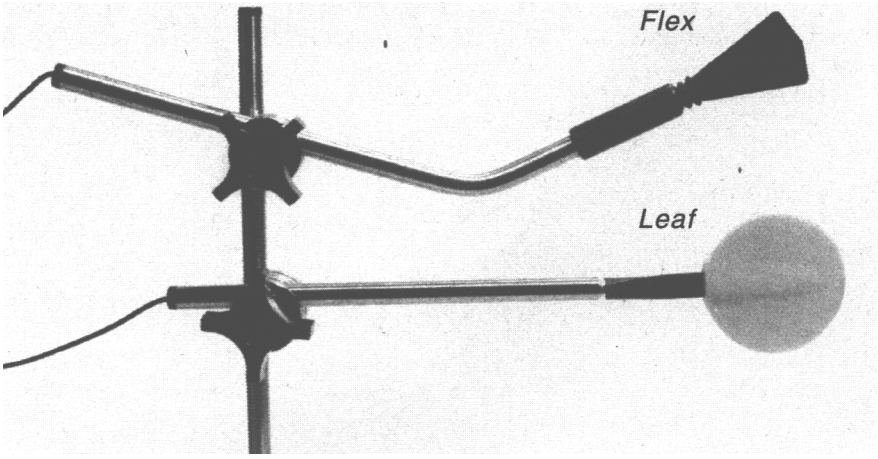

Abbildung 1.6: Beispiele für mechanische Sensoren

1.3.1.2 pneumatische Sensoren

Pneumatische Sensoren werden durch Blasen oder Saugen betätigt. Der Patient hat dabei ein Rohr in der Umgebung seines Mundes, an dessen Ende der Sensor befestigt ist (siehe Abbildung 1.7). Auch hier sind neben den elektrischen Kriterien besondere medizinisch–hygienische Forderungen zu erfüllen.

1.3.1.3 elektronische Sensoren

Die Palette der elektronischen Sensoren ist weit gestreut. Mit Hilfe einer geeigneten Elektronik kann jede eindeutige Veränderung einer physikalischen Größe zur Impulsgewinnung herangezogen werden. Bereits verwirklicht sind unter anderem Systeme zur Erkennung von Geräuschen, Annäherung, Stirnrunzeln, Augenbewegung, Lidbewegung und Kopfneigung.

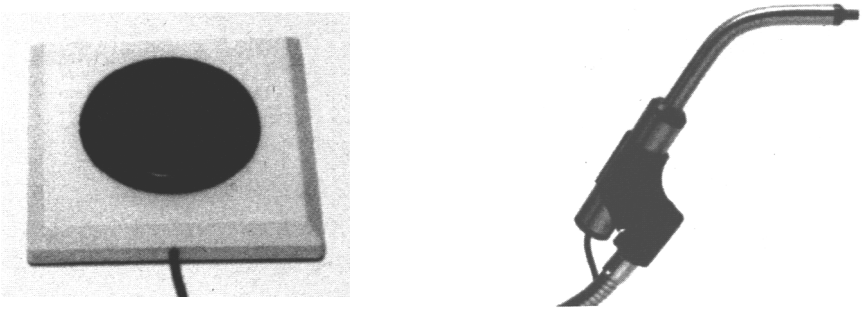

Abbildung 1.7: Beispiele für Sensoren: links mechanischer Sensor — rechts pneumatischer Sensor

1.3.2 Auswertung von Impulsen

Prinzipiell gibt es mehrere Möglichkeiten mit einem Schalter verschiedene „Tasten“ zu simulieren. Bis jetzt wird aber ausschließlich eine Art genutzt — das *Scannen* —, was ein eindeutiger Mangel und einer der Hauptansätze für diese Diplomarbeit ist, die neue Bedienungsmöglichkeiten verwendet, die weiter unten[2] näher erläutert werden sollen.

Das *Scannen* basiert darauf, daß alle zur Auswahl stehenden Aktionen in einem meist einstellbaren Rythmus ausgewählt werden; dies wird optisch zum Beispiel durch ein Aufleuchten eines kleinen Lichtes angezeigt. Mit der einen Taste kann man die derzeit aktive Aktion aktivieren. Bei vielen auszuwählenden Aktionen muß man Gruppen („Untermenüs“) bilden, in denen die verschiedenen Aktionen ausgewählt werden können, um die Zykluszeit (Zeit zwischen der Wiederholung einer bestimmten Aktion) in erträglichem Maß halten zu können. Damit muß zuerst eine Gruppe ausgewählt werden und danach die darin enthaltene Aktion.

1.4 derzeit bestehende Mängel käuflicher Systeme

Bestehende Systeme für Schwerstbehinderte sind stand–alone–Lösungen für spezielle Bereiche[3]. Möchte jedoch ein Behinderter mehrere Systeme gleichzeitig benutzen — zum Beispiel ein Sprachmodul und ein Umweltkontrollgerät —, so war er bisher auf eine betreuende Person angewiesen, die den Sensor zwischen den zwei oder mehreren Systemen umgesteckt hat. Diese Diplomarbeit setzt an diesem großen Mangel an und stellt ein Gerät vor, daß vor die bestehenden Systeme geschaltet wird und die Selektion dem Patienten überläßt, was eine bedeutende Steigerung der Lebensqualität darstellt. Dazu mußten neue Auswahl-

[2] siehe Kapitel 4.1.1 bis 4.1.4 ab Seite 20
[3] siehe Kapitel 1.2 auf Seite 5

techniken entwickelt werden, da die Einzelgeräte keine „Ausstiegsmenüpunkte“ besitzen, um wieder die Kontrolle an das übergeordnete Umschaltgerät zurückzugeben.

Weiters gibt es für manche Bereiche noch überhaupt keine Lösungen, sodaß auch hier Handlungsbedarf besteht. Diese Diplomarbeit ermöglicht es erstmals, ein Mobiltelefon zu steuern, was besonders für Rollstuhlfahrer bei notwendigen Alarm– bzw. Notrufen wichtig ist.

Probleme treten auch bei schon fertig entwickelten Systemen — wie den Computereingabehilfen — auf. Die Bewegung der Mouse kann relativ leicht bewerkstelligt werden, die Mouse–Tastenaktivierung ist aber bei nahezu allen Geräten schwer zu bedienen. Dies gilt besonders für die Aktionen doubleclick und drag–and–drop, da dies eine erhöhte motorische Anforderung an den Patienten stellt. Daher ist in dieser Diplomarbeit ein zusätzlicher Betriebsmodus für die Mouse–Tasten inkludiert, der bestehende Mouse–Bedienungssysteme verbessert.

Kapitel 2

bisherige Forschungen

Das Institut für Biomedizinische Technik und Physik im AKH Wien hat eine Arbeitsgruppe mit dem Titel „Hightech — Technik im Dienste der Behinderten und Betagten“ ins Leben gerufen, die auf dem Gebiet der Technik für Behinderten Forschungen anstellt. Dabei geht es vorallem darum, auf dem Markt befindliche Geräte zu finden und auf bestimmte Kriterien hin zu überprüfen. Besonders wichtig dabei sind die Kriterien der Funktionalität, der Verfügbarkeit, des Services und der Preiswürdigkeit.

Die *Funktionalität* bezieht sich auf Dinge wie Bedienbarkeit, Zuverlässigkeit, Anwenderfreundlichkeit, usw.

Die *Verfügbarkeit* ist eines der ganz großen Probleme auf diesem Gebiet. Leider werden von manchen Geräten nur einige wenige Prototypen entwickelt, die entsprechend wenigen Personen zu gute kommen. Ziel ist es, solche Geräte allen Behinderten zur Verfügung stellen zu können. Weiters wird auch eine weltweite Suche nach geeigneten Hilfsmitteln unternommen, um möglichst alle Herstellerfirmen zu kennen und im Bedarfsfall darauf zurück kommen zu können.

In diesem Zusammenhang ist auch das *Service* wichtig. Kleinere Mängel sollen möglichst schnell vor Ort repariert werden können, um einen längeren Ausfall für den Patienten zu vermeiden, wie es beim Serviceversand ins Ausland passieren kann.

Zuletzt ist auch noch die *Preiswürdigkeit* nicht völlig unerheblich; ein bestimmtes Preis/Leistungsverhältnis muß gewahrt bleiben.

Ein weiterer wichtiger Teil der Arbeitsgruppe besteht in der österreichweit einmaligen dauernden Lehrausstellung, wo sich sowohl Ärzte als auch Betroffene über den derzeitigen Stand der Technik informieren können und Lösungswege aufgezeigt werden.

Bei den Forschungen der Areitsgruppe zeigte sich der derzeit bestehende Mangel bei

der Bedienung von technischen Geräten von Schwerstbehinderten[1], was zur Ausschreibung dieser Diplomarbeit führte.

[1] siehe Kapitel 1.4 auf Seite 12

Kapitel 3

Anforderungsprofil des Gerätes

In diesem Kapitel sollen in allgemeiner Form die Anforderungen an das Gerät dargelegt werden; technische Einzelheiten werden im Pflichtenkatalog[1] definiert.

Die Anforderungen gliedern sich in drei große Bereiche, die bereits oben aufgezeigt wurden[2]:

- Umschaltfunktion für bis zu drei Geräte
- Erweiterung von bestehenden „Mouse–Ersätzen“
- Bedienung eines Mobilfunktelefons

3.1 Umschaltfunktion

Das Gerät soll als Vorschaltgerät dazu dienen, drei „normal käufliche“ Geräte (z.B. Sprachmodul und Umweltkontrollgerät) mit Scanbetrieb und mit nur einem Sensor oder wahlweise mit zwei Sensoren betreiben zu können; dazu müssen geeignete Arbeitsmodis gefunden werden, um die „Auswahl–Clicks“ des Sensors von den durchzuschleifenden Clicks zu unterscheiden, da ein eigener Scanmodus nicht verwendet werden kann, weil dieser unabhängig vom Scanmodus des Zielgerätes wäre und damit unbedienbar wird. Weiters soll die Möglichkeit bestehen, das Gerät mit Hilfe von Infrarotimpulsen zu steuern, wie sie zum Beispiel die „EOG–Brille“ verwendet.

Die Ausgänge 1 und 2 des Gerätes sollen nur die Impulse weiterleiten, der Ausgang 3 soll bei jedem Impuls den Zustand (ein/aus) wechseln (toggle).

Von der Umschaltfunktion soll man mit Hilfe des Sensors in den Mouse–Modus umschalten können, wenn dies nicht bereits durch einen extern anzuschließenden Schalter erledigt

[1] siehe Kapitel 4 ab Seite 19

[2] siehe Kapitel 1.4 auf Seite 12

wird, der z.B. auf einem Rollstuhl fix montiert werden kann und somit automatisch beim Heranfahren an den Computer aktiviert wird.

3.2 Mouse–Funktion

Das Gerät soll die Bedienung von „handelsüblichen“ Mouse–Emulatoren dahingehend erweitern, als daß die Mouse–Tasten durch dieses Gerät ersetzt werden. Die Steuerung soll hier durch die Betätigung eines Clickmodis (click, doubleclick, switch) für die gerade aktive Taste (links, mitte, rechts) erfolgen.

Weiters soll der Mouse–Modus auch durch Scannen bedient werden können; der Infrarotmodus entspricht dabei einem Zweitastenmodus.

Vom Mouse–Modus soll man mit Hilfe des Sensors in den Umschaltmodus wechseln können, wenn dies nicht bereits durch einen extern anzuschließenden Schalter erledigt wird.

3.3 Telefonfunktion

Das Gerät soll mit Hilfe eines handelsüblichen GSM–Mobiltelefon die Möglichkeit bieten, eine gespeicherte oder eine beliebige Telefonnummer anzuwählen und eine Sprechverbindung aufzubauen. Die Steuerung des Telefons ist nur mit dem Infrarotempfänger möglich, um eine zu umständliche Bedienung zu vermeiden. Wenn keine direkte Infrarotsteuerung möglich ist, kann im Umschaltmodus an einen Ausgang eine entsprechende, lernfähige Fernbedienung angeschlossen werden, die eine teilweise Steuerung ermöglicht. Das bedeutet, daß der Telefonmodus sowohl mit dem Mouse–Modus als auch zum Umschaltmodus parallel betrieben werden kann, nur im Infrarotmodus muß eine Umschaltung vorgesehen werden.

Zuletzt ist auch eine Notruffunktion über Telefon vorzusehen, die über einen eigenen Sensoreingang ausgelöst wird.

Kapitel 4

Pflichtenkatalog

In diesem Kapitel werden die technischen Anforderungen an das Gerät detailliert festgehalten und die Bedienungsmodis definiert.

Die bereits erläuterten drei Arbeitsmodis Umschaltfunktion, Mouse und Telefon sollen im folgenden getrennt beschrieben werden, danach wird das Zusammenspiel der drei Funktionen erklärt; zuletzt sollen die Einstellmöglichkeiten und die Stromversorgung festgehalten werden.

4.1 Umschaltfunktion

Zur Steuerung sollen fünf unterschiedliche Steuermodis an Hand der eingesteckten Sensoren automatisch erkannt werden.

Für die Sensoren sollen drei Klinkenbuchsen (E1 bis E3) vorgesehen werden. *E1* dient zur Aufnahme des Schaltsensors im Mod1 und Mod2[1], *E2* dient zur Aufnahme des Schaltsensors im Mod3 und Mod4 und *E3* dient zur Aufnahme des Select–Sensors im Mod2 und Mod4. Durch diese Anordnung ist sichergestellt, daß jeder der Betriebsmodis Mod1 bis Mod4 nur durch die angesteckten Sensoren erkannt werden kann. Zur Erkennung eines angesteckten Sensors werden Klinkenbuchsen mit integrierten Schaltern verwendet.

Auf der Ausgangsseite sollen drei Stereoklinkenbuchsen (A1 bis A3) vorgesehen werden. Die Ausgänge sind Zwecks galvanischer Trennung durch Relais mit Umschaltkontakt zu realisieren. Die Ausgänge können bis auf A1 mittels DIP–Schalter[2] deaktiviert werden, was auch in der Bedienerführung zu beachten ist. Der Ausgang A3 stellt einen Sonderfall

[1] die Definition der Modis 1 bis 4 erfolgt im Kapitel 4.1.1 bis 4.1.4

[2] Eine Autokonfiguration über Klinkenbuchsenschalter ist wegen der galvanischen Trennung zu aufwendig, da eine galvanisch getrennte Spannung zum Test der Schalter benötigt würde.

dar, der als Toggle–Ausgang[3] fungiert.

Im folgenden werden nun die Steuermodis definiert und eine kurze Motivation jedes Moduses gegeben; die Begriffe neben den Modis in Klammern sind interne Kurzbezeichnungen zum besseren Verständnis und zum leichteren Merken der Modis bei Diskussionen.

4.1.1 Mod1 (Single)

Der Mod1 ist der einfachste Modus. Hierbei wird die Bedienung nur über einen einzigen Sensor verwirklicht, der in E1 eingesteckt wird. Dieser Modus ist für Patienten geeignet, die nur **einen** Sensor bedienen können. Der Nachteil gegenüber anderen Modis besteht darin, daß die Dauer des Ausgangsimpulses nicht bestimmbar ist und der Impuls erst nach Deaktivierung des Sensors ausgegeben wird.

Der zeitliche Ablauf wird in Abbildung 4.1 beispielhaft dargestellt. Dabei ist t_{ref} eine

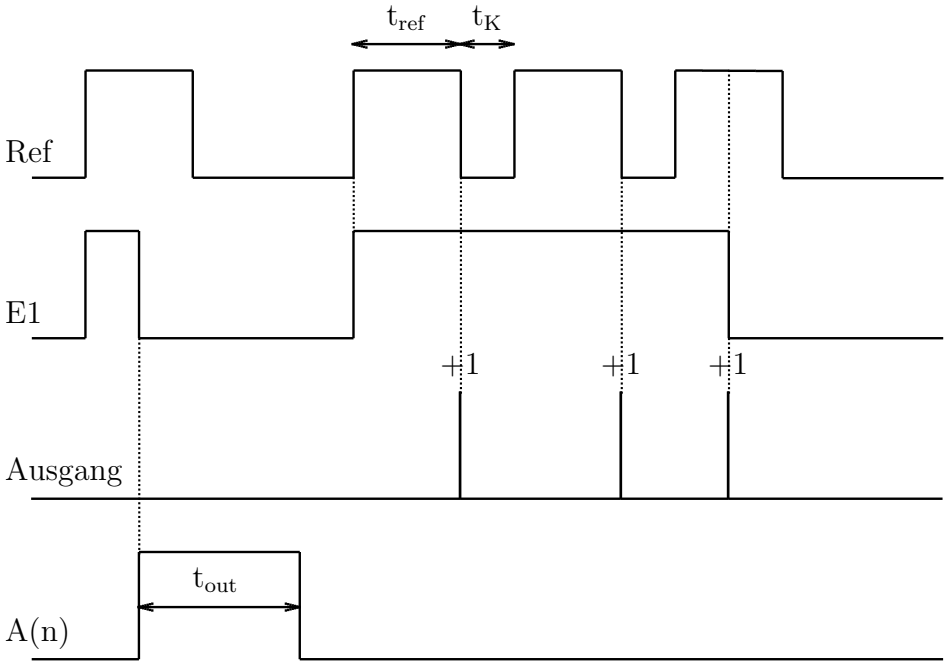

Abbildung 4.1: Zeitplan von Mod1

auf den Patienten abgestimmte Referenzzeit, die die Unterscheidung zwischen „Impuls ausgeben“ und „auf nächsten Ausgang schalten“ ermöglicht. Wird, wie im ersten Teil des Zeitplanes dargestellt, der Sensor an E1 kürzer als die Referenzzeit aktiviert, so wird

[3]Dieser Ausgang schaltet sich nicht solange ein wie der Impuls vorschreibt, sondern wechselt seinen Zustand (ein/aus) bei jedem Impuls.

mit dem Zeitpunkt der Deaktivierung ein Impuls mit der einstellbaren Dauer von t_{out} am derzeit aktivem Ausgang A(n) ausgegeben.

Übersteigt die Zeit der Sensoraktivierung t_{ref}, so wird mit jeder angefangenen Zeit t_{ref} auf den nächsten möglichen Ausgang weitergeschaltet. Nach jeder Referenzzeit schließt sich eine Karrenzzeit t_K an, in der der Patient den Vorgang abbrechen kann. Dies ist besonders im häufigen Fall der Weiterschaltung um 1 notwendig, da die erste Referenzzeit verstrichen sein muß um eine Weiterschaltung zu ermöglichen, eine zweite Referenzzeit aber noch nicht begonnen sein darf, da sonst um 2 Stufen weitergeschaltet werden würde. Die Karrenzzeit t_K kann mit Hilfe zweier DIP–Schalter auf $\frac{1}{8}$, $\frac{1}{4}$, $\frac{1}{2}$ oder die ganze Referenzzeit t_{ref} eingestellt werden.

4.1.2 Mod2 (Single + Select)

Der Mod2 ist eine Weiterentwicklung des Mod1. Dieser benötigt zwei Sensoren, was eine höhere Anforderung an den Patienten stellt. Mit dem einen Sensor (E1) wird der Ausgang nahezu direkt gesteuert, mit dem zweiten (E3) wird der nächste mögliche Ausgang ausgewählt. Der Vorteil gegenüber Mod1 liegt in der Möglichkeit, die Dauer des Ausgangsimpulses direkt mit dem Sensor zu beeinflussen und den nächsten Ausgang einfacher anwählen zu können.

Ein zeitlicher Beispielablauf ist in Abbildung 4.2 dargestellt. t_{ref} ist hier wieder die einstellbare Referenzzeit. Wird der Sensor an E1 kürzer als die Referenzzeit gedrückt, so wird der Impuls bis zum Erreichen der Mindestdauer t_{out} verlängert. Der Unterschied zu Mod1 besteht darin, daß der Impuls bereits mit der Aktivierung des Sensors auf den derzeit aktiven Ausgang A(n) geschaltet wird. Wird der Sensor hingegen länger als die Referenzzeit aktiviert, bleibt der Ausgang A(n) bis zur Deaktivierung des Sensors eingeschaltet.

Wird der Selectsensor in E3 während eines aktivierten Ausgangs betätigt, so wird dieser ignoriert; dies geschieht auch, wenn der Selectsensor gedrückt bleibt, während der Impulssensor in E1 deaktiviert wird. In so einem Fall muß das Gerät auf die Deaktivierung des Selectsensors warten, bis es weitere Eingaben annehmen kann; dieses Verhalten soll Fehlauslösungen einschränken, da manche Patienten sich immer nur auf einen Sensor konzentrieren können und so Fehlauslösungen am anderen Sensor vorkommen würden.

Wird der Selectsensor alleine aktiviert, so wird zum Zeitpunkt der Aktivierung auf den nächsten Ausgang geschaltet und danach alle Eingaben ignoriert, bis der Sensor wieder deaktiviert wird.

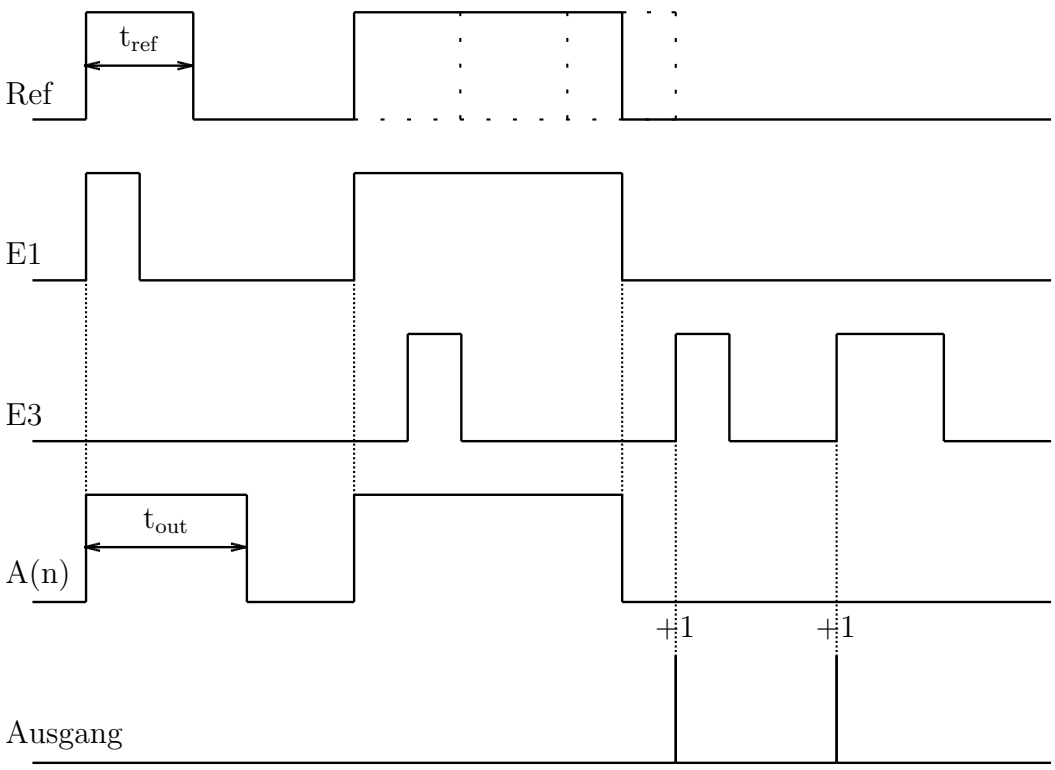

Abbildung 4.2: Zeitplan von Mod2

4.1.3 Mod3 (Double)

Manche Sensoren sind nur in der Lage kurze Impulse konstanter Länge abzugeben; dazu zählen Stirnrunzelschalter und Lidschlagschalter. Für Patienten, die nur einen solchen Sensor benutzen können, wurde der Mod3 entwickelt. Gegenüber den Modis Mod1 und Mod2 hat dieser Modus leider nur Nachteile. Der Ausgangsimpuls wird erst am Ende der Referenzzeit ausgegeben und ist in der Dauer konstant. Mit einer „Doppelaktivierung“ kann immer nur um einen Ausgang weitergeschaltet werden.

Der Zeitplan dieses Modis ist in Abbildung 4.3 dargestellt. t_{ref} ist wieder die einstellba-

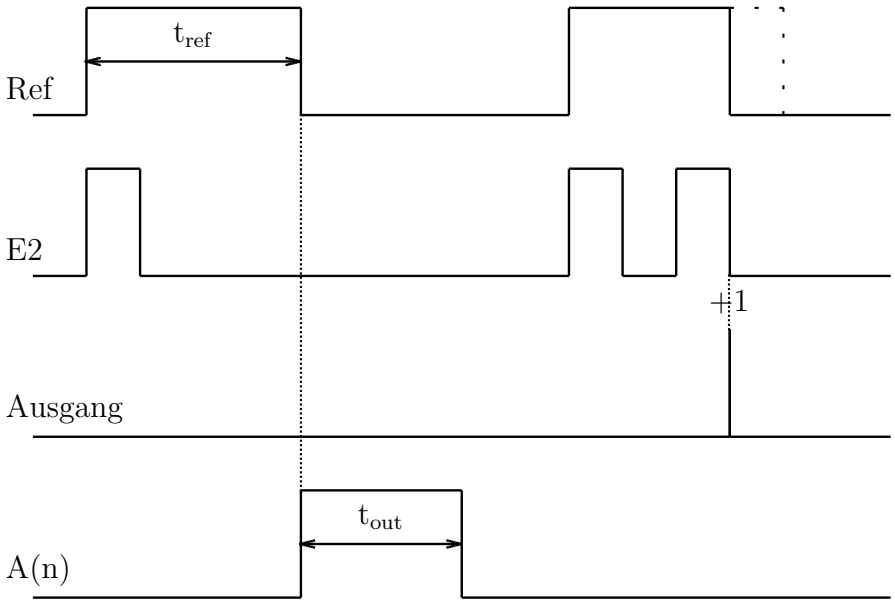

Abbildung 4.3: Zeitplan von Mod3

re Referenzzeit, die die Unterscheidung von Ausgangsimpuls und Weiterschalten möglich macht. Wenn innerhalb der Referenzzeit ein Impuls an E2 gezählt wird, so wird am aktuellen Ausgang A(n) ein Impuls mit der einstellbaren Länge t_{out} ausgegeben.

Wenn innerhalb der Referenzzeit zwei Impulse gezählt werden, wird unmittelbar mit der zweiten Deaktivierung auf den nächsten möglichen Ausgang geschaltet.

4.1.4 Mod4 (Double + Select)

Mod4 ist die Weiterentwicklung des Mod3, setzt aber voraus, daß der Patient zwei Sensoren bedienen kann. Gleichzeitig wird ein Problem bei Lidschlagschaltern gelöst, da Einzelimpulse, wie sie durch unvermeidbares Zwinkern entstehen, ignoriert werden. Gegenüber dem

ähnlichen Mod2 besteht auch beim Mod4 der Nachteil der konstanten Ausgangsimpulsdauer.

Der Zeitplan von Mod4 ist in der Abbildung 4.4 dargestellt. Wenn **ein** Impuls an E2

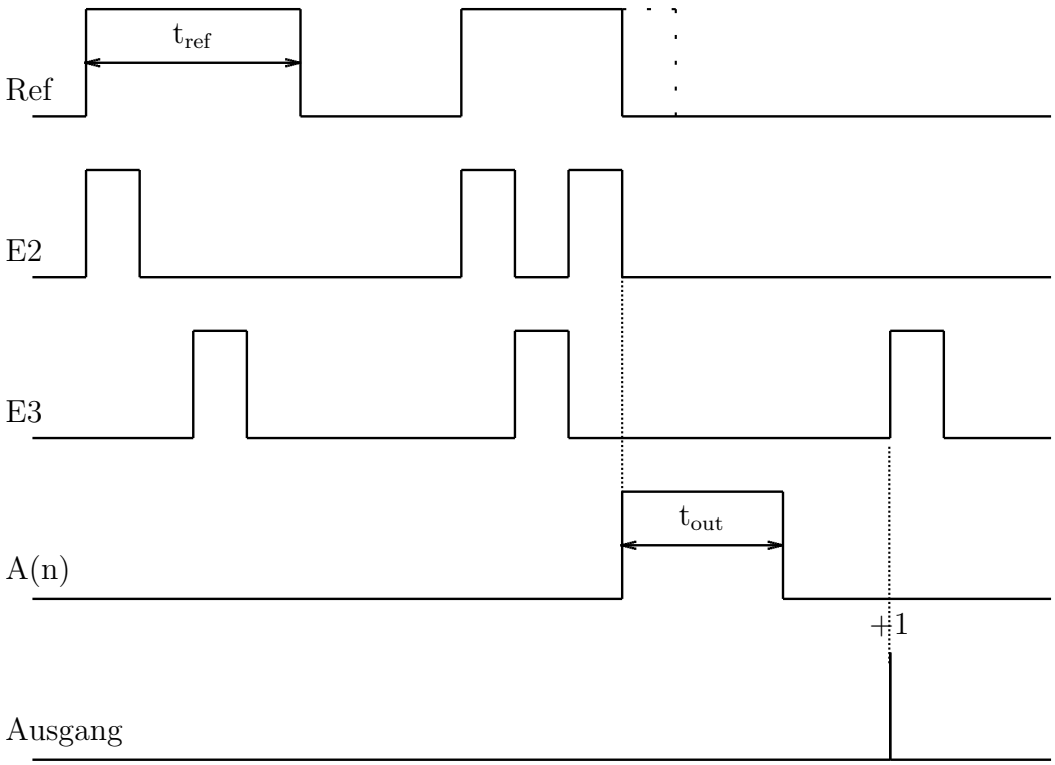

Abbildung 4.4: Zeitplan von Mod4

innerhalb der Referenzzeit t_{ref} gezählt wird, so wird keine Aktion gesetzt, auch ein in dieser Referenzzeit betätigter Selectsensor in E3 wird ignoriert. Sollte hingegen innerhalb der Referenzzeit der Sensor in E2 zweimal betätigt werden, so wird unmittelbar mit Deaktivierung des Sensors am aktuellen Ausgang A(n) ein Impuls mit der einstellbaren Dauer t_{out} ausgegeben; wieder wird ein Impuls am Selectsensor in E3 während der Referenzzeit ignoriert, um — wie im Mod2 — eine Fehlauslösung durch den Patienten zu vermeiden.

Wird der Selectsensor in E3 ohne zeitlichen Zusammenhang mit E1 wie oben beschrieben aktiviert, so wird mit der Betätigung des Selectsensors auf den nächsten möglichen Ausgang geschaltet; danach werden alle Eingaben ignoriert, bis der Sensor in E3 wieder deaktiviert wird.

4.1.5 Mod5 (IR)

Der Mod5 wird aktiviert, wenn kein Sensor in E1–E3 eingesteckt ist und ist vorwiegend im Zusammenhang mit der „EOG–Brille“ definiert worden, sodaß damit eine ideale Zusammenarbeit ermöglicht wird.

Die EOG-Brille kann zwar prinzipiell 16 verschiedene Codes direkt senden, aber dies ist von kaum einem Patienten erlernbar. Daher wird der volle Codeumfang nur für den Telefonmodus verwendet, wobei hier ein Zwischengerät mit Tafel verwendet werden kann.

Für die Umschaltfunktion und die Mousetastenemulation wird daher von nur zwei Codes ausgegangen, die z.B. durch zweimaliges nach rechts oder links Blicken ausgelöst werden.

Die Bedienung im IR–Mode gleicht stark dem Mod2, nur daß die Länge des Ausgangsimpulses immer konstant t_{out} ist, da der übermittelte Code keine „Längeninformation“ enthalten kann. Sofort nach dem Eingang des Impulses C_1 — wie in Abbildung 4.5 dargestellt

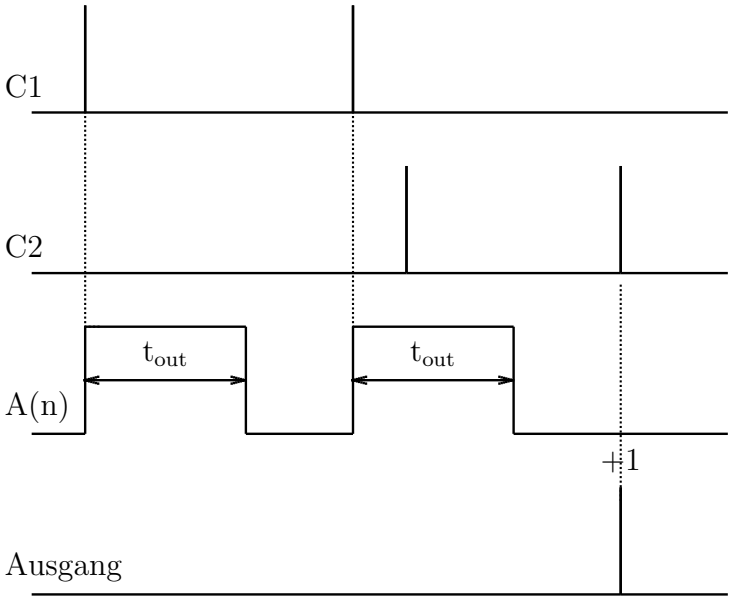

Abbildung 4.5: Zeitplan von Mod5

— wird am aktuell aktiven Ausgang ein Impuls mit der Länge t_{out} ausgegeben. Wird der Code C_2 empfangen, so wird auf den nächsten möglichen Ausgang weitergeschaltet, wenn zu diesem Zeitpunkt kein Ausgangsimpuls abgegeben wird.

4.2 Mouse–Funktion

Das Gerät soll eine Erweiterung bestehender Mouse–Emulatoren darstellen. Dabei werden nur die Tasten ersetzt, da die Bedienung der Tasten im Zusammenhang mit der „Richtungsbewegung“ häufig zu Problemen führt. Die mittlere Mousetaste kann mittels DIP–Schalter deaktiviert werden, da diese in vielen Fällen nicht benötigt wird und so die Bedienung vereinfacht werden kann. Prinzipiell kommen die gleichen Arbeitsmodis wie bei der Umschaltfunktion zum Tragen, wenn auch die Bedienungsoberfläche eine andere ist. Zusätzlich werden noch zwei neue Modis (Mod6 und Mod7) eingeführt, die die Modis Mod1 und Mod3 mit einem Scannen verbinden. Die elektrischen Verbindungen werden über einen Sub D–Stecker bewerkstelligt, der auch die Verbindung zum Telefon ermöglicht.

4.2.1 Bedienungsoberfläche

Im Gegensatz zur Umschaltfunktion, wo mit der Betätigung direkt am Ausgang ein Impuls ausgegeben wurde und mit Select der nächste mögliche Ausgang ausgewählt wurde, wird hier ein zweistufiges System benützt. Zum einfacheren Verständnis ist in Abbildung 4.6 eine mögliche Anordnung skizziert. Auf der linken Seite ist die direkt anwählbare Ebene

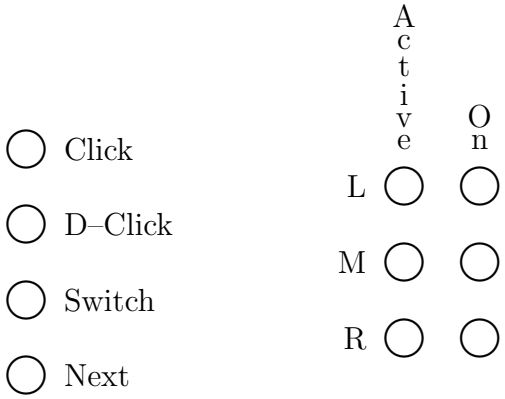

Abbildung 4.6: Skizze zur Erläuterung der Mousetastenbedienung

dargestellt. Der Patient kann mit den verschiedenen Modis (Mod1 bis Mod8) Die Bedienungsart des derzeit aktiven Mouse–Buttons einstellen; der aktive Mousebutton wird durch eine LED in der Spalte *Active* gekennzeichnet. Bei der Aktivierung eines Modis (Click, D–Click oder Switch) wird dieser auf dem aktuellen Mouse–Button ausgegeben. Die nächste Mousetaste wird mit *Next* ausgewählt. Die Spalte *On* ist notwendig, um bei der Bedienung mit Switch die Übersicht zwischen ein/aus zu bewahren, da der Tastenstatus (ein/aus) selbstverständlich auch beim Weiterschalten auf eine andere Mouse–Taste

erhalten bleibt.

4.2.2 Mod6 (Scan–Single) & Mod7 (Scan–Double)

Diese beiden Modis sind die direkte Fortführung der Bedienungsmodis Mod1 (bzw. Mod2) und Mod3 (bzw. Mod4). Wenn die Umschaltfunktion in den genannten Modis betrieben wird und der Scanmodus[4] mittels eines DIP–Schalters aktiviert wird, so gehen die Modis Mod1 und Mod2 bzw. Mod3 und Mod4 direkt in die Modis Mod6 bzw. Mod7 über. Mit dem einzigen Sensor (E3 wird falls angesteckt ignoriert) wird also entweder durch einfaches Betätigen (Mod6) bzw. durch ein zweifaches Betätigen (Mod7) der durch Scannen ausgewählte „Menüpunkt“ ausgeführt.

4.2.3 Mod8 (Scan–IR)

Mod8 wird automatisch ausgewählt, wenn kein Sensor angesteckt ist und der DIP–Schalter für das Scannen eingeschaltet ist. In diesem Mode wird ebenfalls wie bei Mod6 und Mod7 ein Scannen betrieben; trotzdem bleiben beide Codes (C_1 und C_2) aktiv.

4.3 Telefonfunktion

Die Steuerung eines handelsüblichen GSM–Handys ist derzeit leider nicht direkt möglich. Obwohl sehr viele Telefone eine Erweiterungsschnittstelle haben, die theoretisch auch genormt ist, gibt es — wie ich im folgenden erläutern werde — technische Probleme, die ein direktes Steuern unmöglich machen.

Das Hauptproblem liegt in der derzeit sehr rasch voranschreitenden technischen Entwicklung. Leider ist es daher üblich, technische Entwicklungen großer Firmen im Nachhinein zu einer Norm zusammenzufassen. Wenn jedoch mehrere Konzerne verschiedene Entwicklungen hervorgebracht haben, so wird danach in einem Wirtschaftsstreit versucht, die „eigene“ Entwicklung zur Norm zu erheben. Aus diesem Grund ist in der derzeitigen Fassung der GSM–Norm[5] zwar von einer genormten Erweiterungsschnittstelle die Rede, genormt sind aber nur zwei Software–Befehle zum Auslesen und Beschreiben von Registern (siehe dazu den Auszug aus der Norm in Tabelle 4.1). Nicht genormt sind aber die Register und die elektrische Ausführung der Schnittstelle, sodaß jeder Hersteller seine

[4] siehe 1.3.2 auf Seite 12

[5] Genauer: In der GSM 07.06 aus dem „final draft European Telecommunication Standard“ (ETS) von der Special Mobile Group (SMG) Technical Committee (TC) of the European Telecommunications Standards Institute (ETSI)

LIST command

The command ‘LIST’ shall cause the MT to send the current values of one or more parameters from the particular configuration mode.

The use of a LIST command without a specified REGISTER or number of REGISTERS shall cause the MT to send to the TE a complete list of all registers and their current values. List specified parameters.

The use of a LIST message type with a REGISTER or number of REGISTERS shall cause the MT to send to the TE a list of specified registers and their values.

The format for a number of REGISTERS is as follows:
<REGISTER><SPACE><REGISTER><SPACE>...<REGISTER>
where <REGISTER> identifies a particular parameter within the configuration mode.

The CCITT IA5 character SPACE is used as a delimiter to separate one register from another.

SET command

The command ‘SET’ shall allow the TE to set or amend the values of one or more registers representing the parameters for a particular configuration mode.

The <SET COMMAND> consists of a string of CCITT IA5 characters with the eighth bit set to zero. These characters represent the following
<REGISTER>,<VALUE><SPACE><REGISTER>,<VALUE>...
<SPACE><REGISTER>,<VALUE>
where <REGISTER> identifies a particular parameter within the configuration mode and <VALUE> identifies the desired setting of the value for a particular register.

Tabelle 4.1: Auszug aus der GSM-Series 07.06 [4]

eigene Schnittstelle entwickelt hat. Aus wirtschaftlichen Gründen werden diese Schnittstellen aber als Firmengeheimnis betrachtet, da sonst Dritthersteller Erweiterungen zu einem deutlich niedrigeren Preis anbieten könnten als die Originalhersteller selbst.

Ich habe die größten und bekanntesten Hersteller von GSM–Handys kontaktiert und um Ihre Mithilfe bei meiner Diplomarbeit gebeten. Nur wenige Firmen haben auf meine

Anfragen geantwortet und das waren bis auf eine Ausnahme abschlägige Mitteilungen. Die Firma Nokia jedoch hat mich ans Forschungszentrum Tampere in Finnland weiterverwiesen, wo ich zwar nicht die gewünschten Informationen über die Schnittstelle selbst erhalten habe, aber eine andere Lösung aufgezeigt wurde.

Da die Handy–Schnittstelle weitgehend ungenormt ist, wurde von der Industrie von fast jedem Konzern eine Modemerweiterung als Norm–Ersatz herausgebracht, die auf den derzeitigen AT–Kommandos aufbaut; neue Features wie SMS und Adressbuchfunktionen wurden durch neue Befehle inkludiert, die sind aber auch wieder nicht genormt.

An diesem Punkt meiner Forschung stellte sich jedoch ein weiteres Problem heraus. Die österreichische Post unterstützt derzeit nur das „Multinumbering–Verfahren“ und wird nicht vor zwei Jahren auf das „Singlenumbering–Verfahren“ umstellen, wie mir ein zuständiger Entwickler der Firma Kapsch mitteilte. Multinumbering bedeutet aber, daß für jeden Dienst über ein Handy (Sprache, Fax, Modem, ...) eine eigene Telefonnummer benötigt wird. Für den Aktivruf gilt damit, daß bei Benutzung einer Modemkarte zur Wahl eine Fax– bzw. Modemverbindung zustande kommt, was eine Sprachverbindung ausschließt. Fast alle Modemerweiterungen zu GSM–Handys bieten zwar die Funktion „first speak, then fax“ an, dies funktioniert aber nur im Singlenumbering–Verfahren.

Nach weiteren Gesprächen mit dem Forschungszentrum von Nokia konnte ich jedoch auch dieses Problem lösen, was aber eine Einschränkung des zu verwendenden Handys auf ein Nokia 2110 bzw. 8110 mit sich bringt. Diese Geräte bzw. die zugehörige Modemerweiterung ist in der Lage, auch im Multinumbering–Verfahren mit der Erweiterung eine Sprachverbindung aufzubauen.

Daher besteht der Telefonmodus aus einem Nokia 2110 bzw. 8110 GSM–Handy mit zugehöriger Modemerweiterung. Damit ist es möglich mit erweiterten AT–Kommandos eine Sprachverbindung mit Hilfe der gespeicherten Nummern als auch von beliebigen Nummern aufzubauen. Die Ansteuerung erfolgt über eine Standard RS232–Schnittstelle.

Diese Lösung ist **auch** zukunftsorientiert, da nach Umstellung der österreichischen Post eine einfache Anpassung an andere GSM–Handys bzw. Modemerweiterungen möglich ist, da dies nur eine geringfügige Softwareadaption nötig macht; es wäre daher auch vorstellbar, bereits in Betrieb befindliche Geräte ohne großen Aufwand nachzurüsten. Eine totale Abhängigkeit der Telefonfunktion von einer Firma ist daher nicht gegeben!

4.3.1 „normales“ Telefonieren

Die Bedienung des Telefons vom Patienten wird nur über die Infrarotschnittstelle ermöglicht, da alle anderen Methoden zu kompliziert erscheinen. Wenn das Gerät ohne EOG–Brille verwendet wird, kann über ein angeschlossenes Umweltkontrollgerät eine — je

nach Umweltkontrollgerät — teilweise oder vollständige Bedienung innerhalb geschlossener Räume[6] oder mit Hilfe einer direkten Verbindung erreicht werden.

Da bei der EOG–Brille maximal 16 verschiedene Codes zur Verfügung stehen[7], soll die Codetabelle wie folgt definiert werden:

- Die Ziffern 0...9 werden je einem Code zugeordnet
- Ein Code wird zur Wahl von beliebigen Telefonnummern verwendet und im folgenden W genannt.
- Ein Code wird zur Wahl von im Telefon gespeicherten Telefonnummern verwendet und im folgenden M genannt.
- Ein Code wird zur Wahl von in der SIM–Card gespeicherten Telefonnummern verwendet und im folgenden S genannt.
- Ein Code wird zur Wiederwahl der letzten Nummer verwendet und im folgenden R genannt.
- Ein Code wird zum Annehmen eingehender Gespräche und zum Beenden von Gesprächen verwendet und im folgenden H genannt.
- Ein Code wird zur Umschaltung des Telefonmoduses in den Mod5 und umgekehrt verwendet und im folgenden T genannt.

Die Bedienung ist dabei so ausgelegt, daß möglichst wenige Codes gesendet werden müssen.

Zum Aufbau einer Telefonverbindung mit einer beliebigen Nummer muß man W senden, danach die gewünschten Ziffern und zuletzt zum Verbindungsaufbau nochmals W. Sollte beim Eingeben der Nummer ein Fehler aufgetreten sein, so kann man den Vorgang mit H wieder unterbrechen. Während des Gespräches werden alle Codes bis auf H ignoriert um Fehlbedienungen zu minimieren.

Bei der Wahl einer gespeicherten Nummer (auf SIM–Card oder im Telefon) sendet man zu Beginn M für eine im Telefon gespeicherte Nummer bzw. S für eine in der SIM–Card gespeicherte Nummer. Danach kommt eine immer zweistellige Nummer, die den Speicherplatz repräsentiert. Nach der zweiten Ziffer wird die Verbindung automatisch aufgebaut; ein Unterbrechen des Vorganges ist wieder mit H möglich.

[6] zur Reflektion der IR–Strahlen des Umweltkontrollgerätes zum Multiswitch

[7] bei Bedarf mit Verwendung einer Auswahltafel

Da die Wahl einer Telefonnummer, besonders durch Eingabe der vollsändigen Nummer, sehr langwierig sein kann, besteht die Möglichkeit die zuletzt gewählte Nummer mit [R] wiederzuwählen.

Wenn das Telefon einen Anruf meldet (Läuten), so kann mit [H] dieser angenommen werden. Da die Telefonfunktion über eine Modemschnittstellenkarte bewerkstelligt wird, ist es auch durch Setzen des S0–Registers im Modem möglich, daß Anrufe automatisch angenommen werden. Das muß jedoch durch einen PC bewerkstelligt werden und ist **keine** Funktion des Multiswitches.

Ein Gespräch kann immer, egal wie es zustande kam, mit [H] beendet werden.

4.3.2 Notruffunktion

Über den Multiswitch ist eine Notruffunktion über das angeschlossene Handy zu verwirklichen. Dazu gibt es eine eigene Sensorbuchse. Wenn der darin eingesteckte Sensor aktiviert wird, so soll **sofort** die im Telefon gespeicherte Notrufnummer gewählt werden; auf diese Weise könnten auch Pager aktiviert, oder über SMS Internetnachrichten oder Faxe als „Hilferuf“ versendet werden.

4.4 Zusammenarbeit der drei Funktionen

Die oben beschriebenen drei Funktionen werden nicht alle gleichzeitig zur Verfügung gestellt, da sonst die Bedienung zu kompliziert wird. Daher wurde der Funktionsumfang in zwei Bedienungsebenen geteilt.

In der Umweltkontrollfunktion können mit den Bedienungsmoden nur die drei Ausgänge A1–A3 gesteuert werden, eine Mousefunktion ist nicht möglich. Möchte man in den Mouse–Modus gelangen, so gibt es zwei Konfigurationsmöglichkeiten. Entweder wird ein externer weiterer Sensor verwendet, der hardwaremäßig zwischen Mousemodus und Switchmodus umschaltet oder eine softwaremäßige Umschaltung wird gewählt; die Konfiguration erfolgt automatisch mit dem Einstecken des Sensors über einen in die Klinkenbuchse integrierten Schalter.

Softwaremäßig wird im Switchmodus ein weiterer auswählbarer Ausgang hinzugefügt, mit dem man bei Aktivierung in den Mousemode gelangen kann. Im Mousemode wird eine weitere Mouse–Taste hinzugefügt, mit der man in den Switchmodus wechseln kann.

Der Telefonmodus läuft immer parallel zu den beiden oben genannten Umschalt– und Mousemoden, sodaß immer direkt das Telefon bedient werden kann, ohne in einen anderen Modus wechseln zu müssen. Dies ist insbesondere für Patienten ohne EOG–Brille (oder vergleichbaren IR–Sendern) notwendig, um mit Hilfe eines Umweltkontrollgerätes das Te-

lefon bedienen zu können; bei dieser IR–Umgehung ist es natürlich schon notwendig, vom Mousemode in den Switchmode zu wechseln, um das Umweltkontrollgerät als IR–Steuerung benutzen zu können.

Einzig im Mod5 (Infrarotmodus) kann der Telefonmodus aufgrund der begrenzten Codeanzahl nicht parallel betrieben werden, sodaß hier vor der Telefonfunktion der Code [T] gesendet werden muß, um in den Telefonmodus zu gelangen. Während des Gespräches ist es aber möglich, wieder in den Switch– oder Mousemodus zurückzukehren und parallel zum Telefonieren andere Geräte zu bedienen.

4.5 Einstellmöglichkeiten

Wie oben bei den Bedienungsmodis schon angeschnitten, können mehrere charakteristische Zeiten eingestellt werden, um eine bessere Anpassung des Gerätes an den Patienten und die angeschlossenen Geräte zu ermöglichen. Um aber eine Sicherheit gegenüber unbeabsichtigten Veränderungen zu haben, muß vor allen Programmierungen ein DIP–Schalter betätigt werden und ein Klinkenstecker in der entsprechenden Buchse stecken. Es darf immer nur ein einziger Sensor eingesteckt sein, da daran der einzustellende Wert erkannt wird. Manche Werte — wie die Karrenzzeit oder die Scan–Zeit — können zwar ohne Betätigung des DIP–Schalters „Programmieren“ verändert werden, die Änderung wird aber erst bei der nächsten Programmierung (DIP–Schalter „Programmieren“ + eventuell entsprechendem Stecker in einer Klinkenbuchse) wirksam! Im folgenden werden die charakteristischen Zeiten mit den Einstellmöglichkeiten aufgeführt; in Klammer die Bezeichnung, die in den Zeitplänen der Modis verwendet wurden. Alle Zeiten sollen zumindest in einem Bereich von $0.25 \ldots 2$s einstellbar sein.

4.5.1 Scan–Zeit

Im Mod6 und Mod7 wird Scanning betrieben. Die Geschwindigkeit des Scannens muß an den Patienten anpaßbar sein. Dies wird mit Hilfe eines Potentiometers bewerkstelligt, um eine einfache, schnelle aber auch feinfühlige Einstellung zu ermöglichen. Damit eine Einstellung vorgenommen werden kann, muß nur ein einziger Sensor in die hardwaremäßige Umschaltbuchse eingesteckt sein.

4.5.2 Referenz–Zeit

Die Referenzzeit (t_{ref}) ist eine der zentralsten und daher wichtigsten Zeiten bei der Bedienung des Gerätes. Da dies besonders patientenabhängig ist, soll die Einstellung auch vom

Patienten durchgeführt werden können. Diese erfolgt daher mit dem Sensor, der dafür in E1 eingesteckt sein muß.

4.5.3 Karrenz–Zeit

Die Karrenzzeit (t_K) wird mit Hilfe von zwei DIP–Schaltern als Bruchteil der Referenzzeit eingestellt. Dabei sind die vier Kombinationen $\frac{1}{8}$, $\frac{1}{4}$, $\frac{1}{2}$ und $\frac{1}{1}$ möglich. Hier sei nochmals erwähnt, daß die Einstellungen erst bei der Aktivierung (oder bei bereits aktiviertem) „Programmieren“ DIP–Schalter übernommen werden.

4.5.4 Ausgangs–Zeit

Die Ausgangszeit (t_{out}) muß zur Anpassung an die verschiedenen Geräte einstellbar sein. Diese wird mit Hilfe eines in E2 eingesteckten Sensors eingestellt.

4.5.5 Fixzeiten

Einige Zeiten werden fix einprogrammiert und können daher nur im ROM bei der Programmierung verändert werden, oder softwaremäßig über einen an die serielle Schnittstelle angeschlossenen PC[8] Diese Zeiten werden deshalb nicht am Gerät einstellbar gemacht, da Sie für alle Bedingungen ungefähr gleich groß sind, oder mit anderen einfacheren Mittel (z.B. die Clickzeit über Windows^{TM})einstellbar sind.

Prell–Zeit: Die Entprellung der Sensoren wird softwaremäßig durchgeführt. Die dafür notwenige Entprellzeit sollte für alle Sensoren ein bestimmtes Maß nicht übersteigen.

Click–Zeit: Die Dauer eines Clicks für eine Mouse ist konstant.

Doubleclick–Zeit: Die Dauer zwischen zwei Clicks kann über den Mousetreiber leichter eingestellt werden; daher reicht ein fixer Durchschnittswert.

4.6 Stromversorgung

Die Stromversorgung erfolgt extern mit mindestens 9V Gleichspannung oder mindestens 12V Wechselspannung und ist gegen Überspannung (in begrenztem Ausmaß) zu sichern. Sollte die externe Stromversorgung abgeschaltet oder getrennt werden, so soll der Prozessor

[8] Aus Gründen die später erläutert werden, wurde zusätzlich eine multifunktionale serielle Schnittstelle eingeführt. Über diese (siehe Kaptitel 8.3 ab Seite 69) ist eine Änderung jeder Konstante möglich; diese Änderungen gehen jedoch nach einem Reset verloren.

in den Power–Down–Modus übergeführt werden und der Speicher wird von einer Lithium–Backup–Batterie erhalten, damit alle Einstellungen nicht verloren gehen. Die Batterie kann ohne Speicherverlust gewechselt werden, wenn die externe Versorgung eingesteckt und diese ausreichend ist.

4.7 Reset

Da bei einem Prozessor ein Fehler niemals ausgeschlossen werden kann, soll es die Möglichkeit geben, das System durch einen Generalreset völlig zurückzusetzen; dabei gehen auch alle Einstellungen verloren. Dies wird durch Abstecken der externen Versorgung (Power-Down–Modus) und anschließendem Wiederanstecken mit gedrückt gehaltenem Sensor in E1 und in E2 bewerkstelligt.

4.8 Zusammenfassung

Nach allen oben genannten Anforderungen an das Gerät könnte ein mögliches (nicht verpflichtendes) Layout für die Bedienungsoberfläche bzw. Anschlüsse wie in Abbildung 4.7 aussehen. Links oben ist der Infrarotsensor für den Mod5 dargestellt. Darunter ist die bereits bekannte Anordnung zur Mouse–Tastenemulation zu sehen; neu hinzugekommen ist — wie im Kapitel 4.4 erläutert — eine LED für die Abschaltung des Mousemodes. Rechts oben ist die Funktionsgruppe für den Switchmode dargestellt, wobei auch hier die LED zum Einschalten des Mousemodes hinzugefügt wurde. Daneben an der Außenseite sind die den LEDs zugeordneten Ausgangsbuchsen erkennbar. Am unteren Ende des Gerätes kann man die beiden Stecker für die externe Versorgung und für die Mousetasten bzw. serielle Schnittstelle sehen. Auf der linken Außenseite sind die Klinkenbuchsen für den Schalter für hardwaremäßiges Umschalten in den Mousemode, für die Bedienungssensoren und für den Notrufsensor angebracht; auf der rechten Seite befinden sich die Klinkenbuchsen für die drei Ausgänge.

Weiters stellen sich die DIP–Schalter nach obiger Beschreibung wie in Tabelle 4.2 dar. Mit den Positionen *2* und *3* können die Ausgänge A3 und A2 abgeschaltet werden, wenn dort keine Geräte angeschlossen sind; damit ist eine deutliche Erleichterung bei der Bedienung gewährleistet. Weiters kann der Mousemode mit der Position *4* gänzlich deaktiviert werden, um langwierige Bedienungserklärungen für den Mousemode einzusparen, wenn der Patient keine solche Funktion benötigt; in einem solchen Fall kann nicht in den Mousemode geschaltet werden und die entsprechenden LED können nicht aktiviert werden. Mit dem Schalter auf Position *5* kann das Scannen im Mousemode (Mod6, Mod7 & Mod8) aktiviert werden. Mit der Position *6* kann die mittlere Mousetaste deaktiviert werden, da diese nur

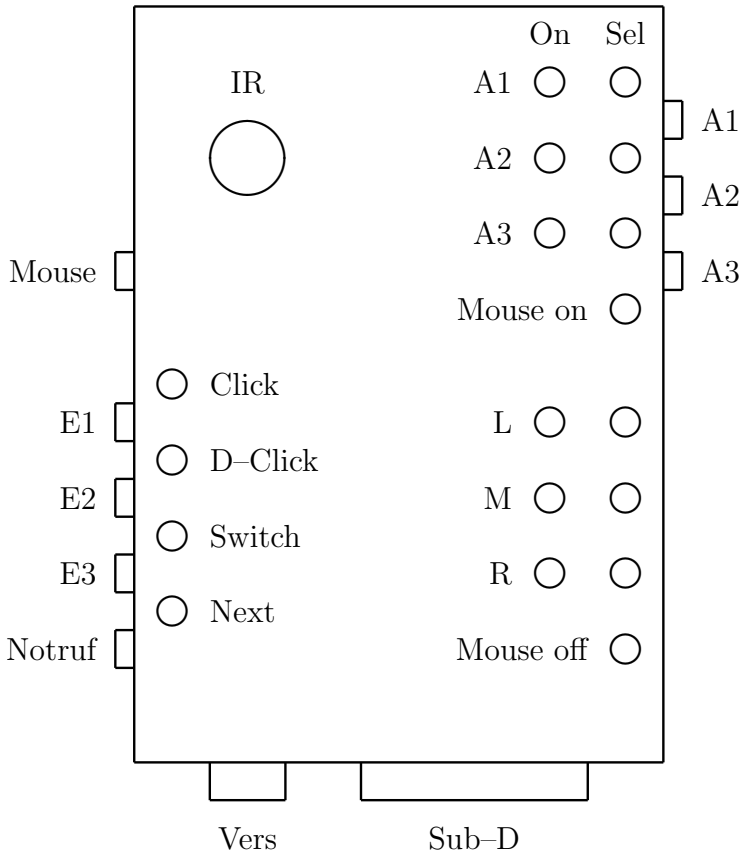

Abbildung 4.7: Beispiel–Layout des Gerätes als Zusammenfassung

0 1	Karrenzzeit-Einstellung als Verhältnis der Referenzzeit	
2	out 3 on/off	falls das 3. Gerät nicht angeschlossen ist
3	out 2 on/off	falls das 2. Gerät nicht angeschlossen ist
4	mouse on/off	Mousemode insgesamt ein/aus
5	scan on/off	schaltet Scan-Mode im Mousemode ein/aus
6	MouseMitte on/off	schaltet den mittleren Mousebutton ein/aus
7	progable on/off	schaltet die Programmiermöglichkeit ein/aus

Tabelle 4.2: Belegung der DIP–Schalter

selten benötigt wird und dadurch eine weitere Vereinfachung der Bedienung möglich wird. Mit dem Switch auf Position *7* wird der Programmiermodus aktiviert, der sich, wie im

Kapitel 4.5 definiert, verhält. Die Positionen *0* und *1* werden zur Einstellung der Karrenzzeit — ebenfalls im Kapitel 4.5 erläutert — verwendet. Insgesamt kann damit ein 8–fach DIP–Schalter verwendet werden.

Teil II

theoretische Ausführung

Kapitel 5

Prozessorauswahl

Zur Verwirklichung des Projektes mußte ein entsprechender Prozessor ausgewählt werden, der für dieses Gerät optimal geeignet ist. Zunächst werde ich daher im folgenden einige Kriterien aufführen, danach zwei mögliche gängige Prozessoren kurz vorstellen und zuletzt meine Wahl begründen.

5.1 Auswahlkriterien

Um eine objektive Auswahl aus der Vielfalt der im Handel erhältlichen Prozessoren zu gewährleisten, werde ich nun einige Kriterien aufzählen und ihre Bedeutung für dieses Gerät im speziellen erläutern. Diese Liste ist sicher nicht vollständig, reicht aber aus, um in meinem Fall eine eindeutige Entscheidung fällen zu können.

Versorgungsspannung: Der Multiswitch wird von einer externen Batterie (oder auch von einem Netzgerät) versorgt. Daher sollte der Verbrauch so gering wie möglich sein. Besonders wichtig ist es, ob der Prozessor einen Sleep– oder Power–Down–Modus hat, bei dem nur der Speicher erhalten wird und der Verbrauch auf den geringstmöglichen Wert sinkt. Dies ist notwendig, da bei abgesteckter externer Versorgung, der Speicherinhalt durch eine interne Lithiumbatterie erhalten werden soll.

Arbeitsgeschwindigkeit: Die Arbeitsgeschwindigkeit ist bei diesem Projekt nicht maßgebend, da — aus Sicht eines Prozessors — alle Vorgänge sehr langsam ablaufen.

Anzahl der I/O–Ports: Die Anzahl der benötigten I/O–Ports ist beim Multiswitch sehr groß, sodaß dies eine stark einschränkende Forderung ist. Sollte die Anzahl der inkludierten I/O–Ports nicht ausreichend sein, so ist die Frage wichtig, wie und ob der Prozessor ausbaufähig ist.

externer Schaltungsaufwand: Da das Gerät in einem relativ kleinen Gehäuse untergebracht werden soll, ist der externe Schaltungsaufwand wichtig. Dieser kann durch notwendige externe Speicher, Latches oder Multiplexer verursacht werden.

notwendige Hardware: Das Gerät stellt einige notwendige Hardwareanforderungen, die zumindest erfüllt werden müssen; z.B. serielle Schnittstelle, Interruptfähigkeit, Zähler, usw.

Prozessorarchitektur: Für die Programmierung ist es nicht unwichtig, ob ein Prozessor z.B. ein RISC-Prozessor ist, wie die Befehlsstruktur aussieht, wie die Hardware angesprochen wird, usw.

Gehäuse: Wie beim externen Schaltungsaufwand ist die Bauform des Gehäuses für den Platzbedarf wichtig.

5.2 Möglichkeit I: PIC16C74

Der PIC16C74 von der Firma Microchip ist ein 8 Bit Singlechip-Prozessor. Der inter-

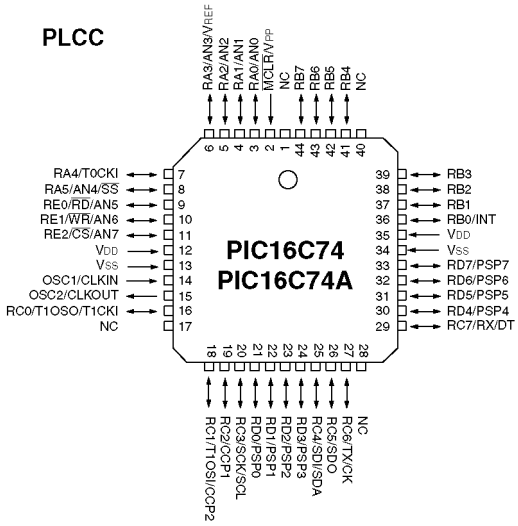

Abbildung 5.1: Darstellung des Gehäuses des PIC16C74 [10]

ne Aufbau ähnelt einer RISC–Struktur, wodurch auch die Befehlsanzahl auf 35 begrenzt ist, die aber alle (bis auf die Jump–Befehle) in einem einzigen Zyklus ausgeführt werden können. Die Ausführungsgeschwindigkeit ist daher für einen Singlechipprozessor bei der 20MHz–Version enorm, sodaß ein Befehl innerhalb von 200ns ausgeführt wird.

Die weiteren Features im Überblick:

- Der Prozessor unterstützt einen externen Interrupt; alle Interrupts haben die gleiche Priorität.
- Der Stack ist, wie bei RISC–Prozessoren üblich, hardwaremäßig ausgeführt und bis zu einer Tiefe von 8 verschachtelbar.
- Es ist ein 4kB ROM on chip, der einmalprogrammierbar und gegen Auslesen schützbar ist.
- 193 Byte RAM (inklusive der Special–Register)
- Der Stromverbrauch liegt im Operationsmodus bei 2mA bei einer Betriebsfrequenz von 4MHz.
- Unterstützung eines Sleep–Modes, bei dem der Verbrauch auf 1μA sinkt.
- 8 gemultiplexte 8 Bit–Analogeingänge
- Spannungsversorgung von 3 bis 6V
- DIL 40 oder PLCC 44 Gehäuse (wie in Abbildung 5.1 dargestellt)
- 3 Timer/Counter
- synchroner und asynchroner serieller Port
- passiver (extern steuerbarer) 8 Bit Parallel–Slave–Port
- 33 mit anderer Hardware geteilte I/O–Pins

5.3 Möglichkeit II: 80C562

Der 80C562 von der Firma Philips ist ein 8 Bit Singlechip–Prozessor. Der Prozessor basiert auf einem 80C51 (von Intel), der mit zusätzlicher Hardware erweitert wurde. Hier ist zu erwähnen, daß der Prozessor 80C562 einen „großen Bruder“, den 80C552 hat, der einige wenige Features mehr hat; diese werden im folgenden mit (♣) gekennzeichnet. Die Struktur des Prozessors erlaubt eine einfache Programmierung, die durch eine große Anzahl (111) vielseitiger Befehle erreicht wird. Neben den üblichen Grundbefehlen für Datentransfer, Entscheidungen usw. sind auch solche für den inkludierten arithmetischen und Bit–Prozessor enthalten; selbst BCD Ein– und Ausgaben sind damit direkt möglich. Bei einer 12MHz–Version werden so 58% der Befehle innerhalb 1μs (1 Zyklus) ausgeführt.

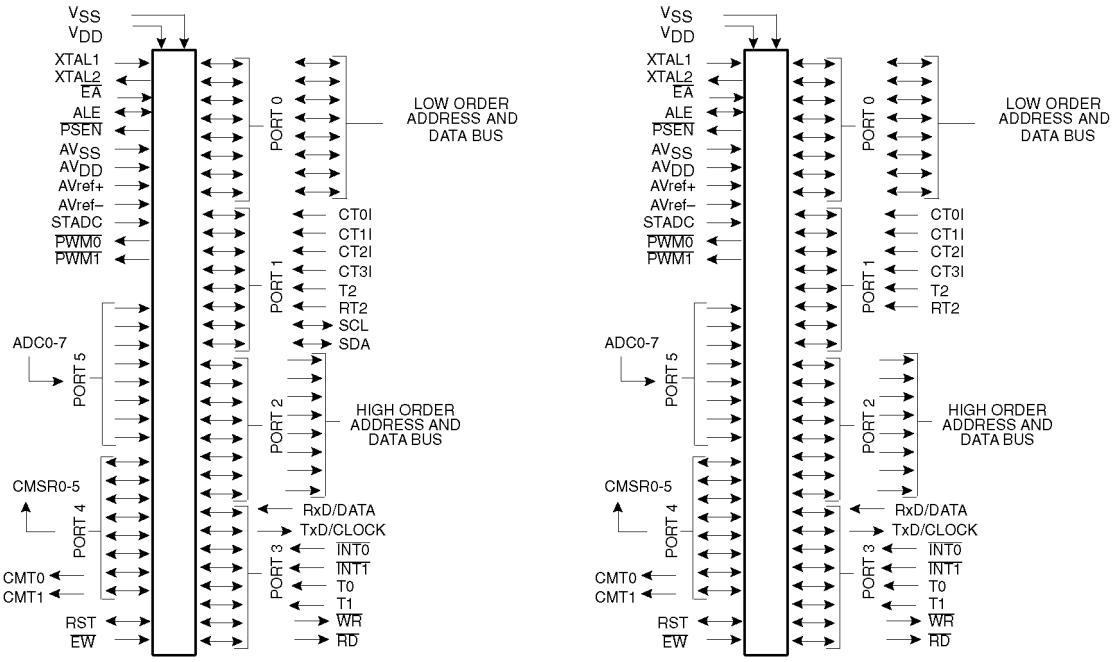

Abbildung 5.2: logische Darstellungen der Prozessoren 80C552 (links) und 80C562 (rechts) [1]

- Der Prozessor unterstützt zwei externe Interrupts; die Interrupts haben zwei Prioritätsstufen.
- Der Stack ist softwaremäßig realisiert und ist nur durch den verwendbaren internen Speicher von 256 Bytes begrenzt.
- Es ist optional ein 8kB ROM on chip, der firmenspezifisch maskenprogrammierbar oder als OTP–Version erhältlich ist; 64kB externer ROM ist zusätzlich möglich. Die EPROM–Version (mit Fenster zum Löschen) ist nur für Entwicklungen empfehlenswert, da sie enorm teuer ist.
- 256 Byte RAM (exklusive der Spezial–Register) intern, extern noch zusätzlich 64kB möglich.
- Der Stromverbrauch liegt im Operationsmodus maximal bei 30mA bei einer Betriebsfrequenz von 12MHz.
- Unterstützung eines Idle– und eines Power–Down–Modus, bei dem der Verbrauch auf 7mA (bei 12 MHz) bzw. 50μA sinkt.
- 8 gemultiplexte 8 Bit (10 Bit ♣) Analogeingänge

- Spannungsversorgung von 4 bis 6V; 2 bis 6V im Power–Down–Modus
- PLCC 68 Gehäuse[1]
- 3 Timer/Counter
- synchroner (♣) und asynchroner serieller Port
- vollständiges aktives RAM/ROM Management
- 48 mit anderer Hardware geteilte I/O–Pins

5.4 Gegenüberstellung der Prozessoren

5.4.1 Vorteile des PIC16C74 gegenüber dem 80C562

Der PIC16C74 hat den Vorteil eines größeren Versorgungsspannungsbereiches. Damit wäre es — zumindest theoretisch möglich — den Prozessor mit einer 3V Lithiumbatterie zu betreiben. Da aber durch die Versorgungsspannungsumschaltung mit einem Spannungsverlust zu rechnen ist, müßte eine Sonderversion der Lithiumbatterie verwendet werden, die eine Spannung von z.B. 3.6V zur Verfügung stellt. Der Verbrauch scheint auch niedriger als der des 80C562 zu sein, obwohl die Angaben irreführend sind, da bei CMOS–Systemen der Verbrauch stark mit der Taktrate gekoppelt ist und die Angaben des PIC16C74 nur bei 4MHz zu finden waren. Trotz der fehlenden Angaben kann man von einem Verbrauchsvorteil von mindestens 100% ausgehen. Noch drastischer ist der Unterschied im heruntergefahrenen Modus (Sleep–Modus, Power–Down–Modus). Hier steht das Verhältnis mit 1:50 für den PIC16C74. Bei einer durchschnittlichen Backupbatterie in Knopfform kann der 80C562 ungefähr ein Jahr erhalten werden, der PIC16C74 länger als die Lagerdauer der Batterie.

Die Arbeitsgeschwindigkeit ist beim PIC16C74 sicher größer als die des „Konkurrenten“. Wie ich aber schon im Kapitel 5.1 erläutert habe, ist dies in diesem Projekt unerheblich.

Der externe Schaltungsaufwand wird durch den internen ROM zunächst gegenüber dem 80C562 verringert, wobei jedoch auch dieser Vorteil durch die OTP–Version des 80C562 verschwindet. Wie ich jedoch im Kapitel 5.4.2 zeigen werde, ist dies der einzige (mögliche) Vorteil in diesem Bereich.

Das Gehäuse ist geringfügig kleiner, da es sich um ein PLCC 44 (gegenüber PLCC 68) handelt.

[1] Das Logikdiagramm ist in Abbildung 5.2 dargestellt.

5.4.2 Vorteile des 80C562 gegenüber dem PIC16C74

Der Nachteil des kleineren Versorgungsspannungsbereiches des 80C562 wird teilweise wieder relativiert, da im Power–Down–Modus die Spannung bis auf 2V abgesenkt werden darf, sodaß auch eine schon sehr leer Batterie noch ausreicht.

Die Prozessorarchitektur ist zwar prinzipiell langsamer, was aber in diesem Fall kein Hindernis darstellt. Für den Programmierer werden viele beim PIC16C74 sehr schwer lösbare Probleme zu einfachsten Aufgaben. Die Nachteile die man für eine höhere Ausführungsgeschwindigkeit erkauft, sind bei diesem Projekt sehr störend und die höhere Ausführungsgeschwindigkeit wird nicht benötigt.

Die notwendige externe Hardware des 80C562 ist insgesamt geringer als die des PICs. Der Philipsprozessor benötigt zwar einen externen ROM (oder nach erfolgter Entwicklung einen internen EPROM), hat aber sehr viel mehr I/O–Pins, die eine deutliche Erleichterung bringen. Der größte Nachteil des PIC16C74 besteht in der völlig auf Singlechiparchitektur ausgerichteten Hardware. Solange die onchip–Hardware ausreicht, ist das System einfach zu verwenden; leider wurde jedoch für einen weiteren Ausbau keine unterstützenden Maßnahmen gesetzt. Da für den Multiswitch ca. 46 I/O–Pins notwendig sind, müssen zumindest 13 Pins beim PIC16C74 hinzugefügt werden. Dies ist aber in keinster Weise mit der bereitgestellten Hardware möglich. Es bliebe nur die Variante, z.B. 3 Pins an einen Demultiplexer anzuschließen, der damit 8 Ausgänge schaffen würde. Auf diese Weise müßten zumindest 3 solcher Multiplexer eingesetzt werden; der dabei auftretende Codier/Decodieraufwand mit einer RISC–Struktur ist jedoch beträchtlich. Der 80C562 unterstützt ein volles externes Interface für RAM und ROM. Daher ist es ein Leichtes, zur Erweiterung Latches memorymapped in das System einzubinden. Weiters stehen schon von vornherein sehr viel mehr I/O–Pins zur Verfügung, sodaß dies einen deutlich vereinfachten Hardwareaufwand bedeutet. Der externe ROM des Philipsprozessors birgt auch Vorteile in sich. Da das Programm zu einem sehr großen Teil aus Entscheidungen und Ein/Ausgabeprozessen bestehen wird, ist es sehr speicherintensiv. Es ist daher zur Zeit der Prozessorwahl nicht sicher, ob man — besonders bei einer RISC–Struktur, wo ein bedingter Sprung 3 bis 4 Befehle benötigt — mit dem internen ROM des PICs von 4kB auskommt; eine Erweiterung ist aber dort **nicht** vorgesehen, was zu einem Scheitern des Projektes führen würde. Der 80C562 kann mit einem externen ROM von 64kB ausgestattet werden, was bei der mächtigeren Befehlsauswahl zudem noch ein um einiges kompliziertes Programm ermöglicht.

Zuletzt ist der Philipsprozessor durch seine Ausbaufähigkeiten sehr zukunftsorientiert, da Erweiterungen an Programm und/oder Hardware einfach möglich sind. Auch hat der externe ROM den Vorteil, das dieser von weiter entfernten Servicestellen gebrannt und ersetzt werden kann, was bei PICs nicht möglich ist, da er eine eigene Programmierumgebung

benötigt und nur einmal gebrannt werden kann; eine Softwareerneuerung kann somit nur durch einen neuen Prozessor ermöglicht werden. Sollte jedoch eine höhere Packungsdichte am Print oder eine größere Stückzahl von Nöten sein, so kann immer noch mit geringem Aufwand auf die OTP–Version umgestiegen werden, was in der derzeitigen Ausbaustufe dazu führen würde, daß keine externen Latches notwendig wären.

5.4.3 „Sieger“ 80C562

Aus den Ausführungen in Kapitel 5.4.1 und 5.4.2 ist eindeutig zu erkennen, daß der PIC16C74 ein guter Prozessor ist, solange man nicht mehr als die vorhandene Hardware benötigt. Da aber in diesem Projekt deutlich mehr I/O–Pins und eventuell mehr ROM benötigt wird, fällt die Wahl auf den flexibleren 80C562. Aus wirtschaftlichen Gründen kann auch bei diesem Modell auf den völlig kompatiblen Typen 80C552 zurückgegriffen werden.

Kapitel 6

Prozessorarchitektur

Wie bereits oben[1] kurz angeschnitten, ist die Prozessorarchitektur des 80C562[2] so gestaltet, daß sie diesem Projekt entgegenkommt; einfache Programmierung durch komplexe Befehle, fast alle benötigten Hardwarekomponenten on chip, aber trotzdem vielfache Erweiterungsmöglichkeiten für zukünftige Verbesserungen.

Im folgenden soll ein kurzer einführender Überblick über den verwendeten Prozessor gegeben werden, der aber auf keinen Fall das zugehörige Handbuch bzw. die Datenblätter ersetzen will, sondern nur eine mit Absicht sehr kurz gehaltene allgemeine Einführung für den „schnellen Leser“ ohne Hintergrundinformation darstellt. Zum genauen Studium sei hier auf das zugehörige User Manual [9] bzw. das allgemeine Datenbuch der 80C51–Reihe [1] der Firma Philips verwiesen; die folgenden Bilder dieses Kapitels sind auch diesen Büchern entnommen worden.

Weitergehende Informationen über den Prozessor sind in den entsprechenden Kapiteln der Hard– und Softwareentwicklung an geeigneter Stelle zu finden.

6.1 Allgemeines

Der 80C562 ist ein 8–Bit Prozessor, dessen Speicher in Bytes organisiert ist; daher sind auch alle Befehle für Byte– oder Bit– Manipulationen ausgerichtet. Als Adressierungsarten stehen register–implizit, direkt, indirekt und immidiate zur Verfügung, wobei aber nicht alle Befehle mit allen vier Adressierungsarten verwendet werden können.

In der Abbildung 6.1 ist eine Übersicht der onchip–Hardware zu sehen. Auf der linken

[1] siehe Kapitel 5.3 auf Seite 41

[2] Wie schon erläutert kann auch an Stelle des 80C562 der „große Bruder“ 80C552 verwendet werden, der einige wenige Features mehr hat. Im folgenden sind daher beide Prozessoren gemeint, wenn nicht ausdrücklich auf eine der beiden Versionen hingewiesen wird; die Diagramme sind für den 80C552 gedacht,

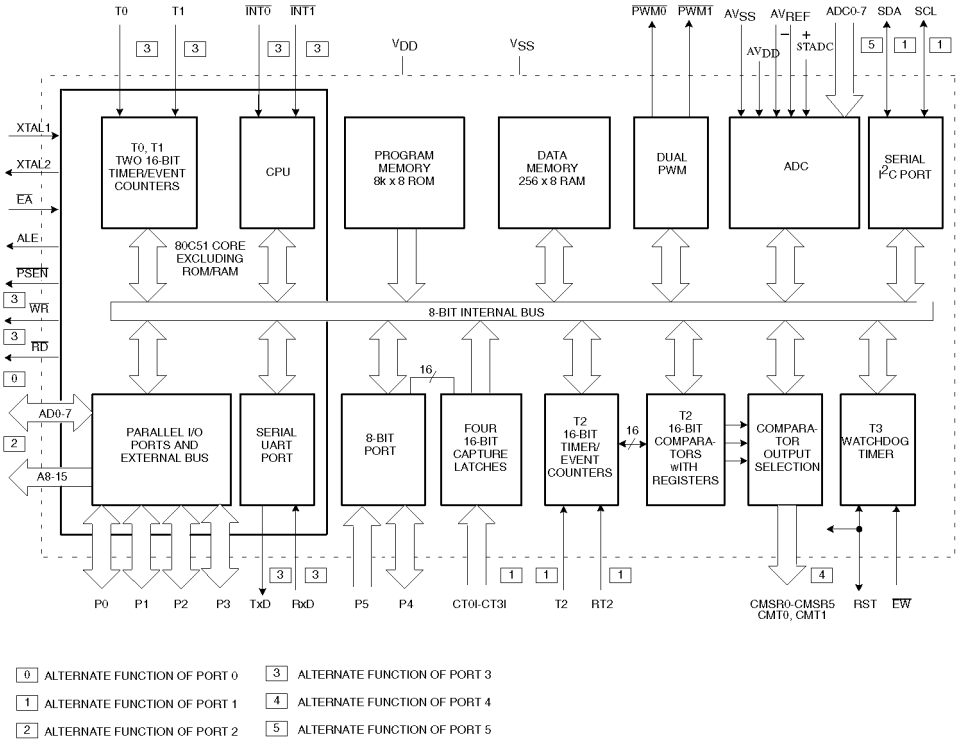

Abbildung 6.1: Blockdiagramm des 80C552 [1]

Seite ist der Teil des 80C51 Grundbausteins umrandet dargestellt, der Rest stellt die zusätzlichen Bauteile dieser Version dar. Einige der Hardwarekomponenten (z.B. I^2C[3], PWM, Comparator, ...) werden derzeit nicht verwendet, stören aber natürlich auch nicht.

6.2 Memory–Organisation

Der Prozessor verfügt über eine Möglichkeit zur Steuerung von 64K externem Daten–Memory, 64K Programm–Memory, internen 256 Byte Daten–Memory und überlappenden 128 Byte spezielle Steuerregister; dabei können bis zu 8K des Programm–Memorys onchip als maskenprogrammierbarer (83C562) oder einmalprogrammierbarer (87C562) ROM ausgeführt sein. Weiters sind 256 Bit des internen Datenspeichers bitadressierbar. Im Bild 6.2 ist die Speicher–Organisation graphisch dargestellt. Mit einem extern zu beschaltenden Pin ($\overline{EA}$) kann gewählt werden, ob die unteren 8K des Programmspeichers intern oder extern ausgeführt sind.

---

die entsprechenden Diagramme des 80C562 sind im Anhang A auf Seite 201 zu finden.

[3] beim 80C562 fehlt der I^2C–Port, siehe Anhang A ab Seite 201

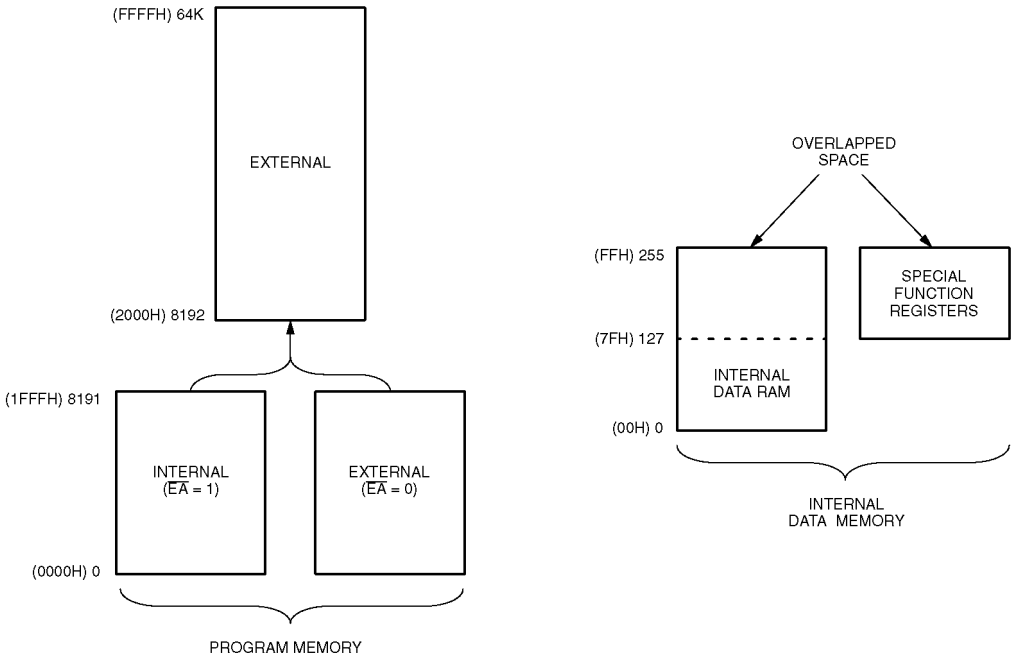

Abbildung 6.2: Darstellungen zur Erläuterung der Speicher–Organisation [1]

Beim Datenspeicher ist der interne und der externe Memory stark getrennt. Externen Speicher kann man nur über `MOVX` Befehle ansprechen. Dabei wird über einen 16-Bit Datenzeiger (DPTR) über indirekte Adressierung zugegriffen; zusätzlich kann eine 256-Byte große „Seite“ mit Hilfe der Register R0 und R1 angesprochen werden, wodurch der Port 2 weiterhin benützt werden kann.

Der interne Speicher ist in drei Teile getrennt. Der Speicherbereich von 0 bis 127 Byte ist allgemeiner Speicher der direkt und indirekt adressiert werden kann, wie es in der Abbildung 6.3 dargestellt ist; darin enthalten sind jedoch auch die Speicherbereiche der 4 Registerbänke und ein Teil der direkten Bitadressen. Im Bereich von 128 bis 255 Byte ist die direkte Adressierung für die speziellen Hardwareregister vorbehalten, mit der indirekten Adressierung kann der obere Bereich des Arbeitsspeichers angesprochen werden. Ein kleiner Teil des allgemeinen Datenspeichers sowie fast alle Spezialregister sind durch den Bitprozessor direkt bitadressierbar, was eine sehr einfache Programmierung der Kontroll– und Portbits als auch der entsprechenden Datenbytes ermöglicht.

Der Stack kann überall im internen Speicher abgelegt werden und ist nur durch den vorhandenen Speicher begrenzt.

Als Arbeitsspeicher steht ein allgemeiner Accumulator (A) und ein B–Register für die Multiplikations– und Divisionsbefehle zur Verfügung. Weiters gibt es noch einige Register für die indirekte Adressierung und den Stack. Für manche Befehle sind weitere Arbeits-

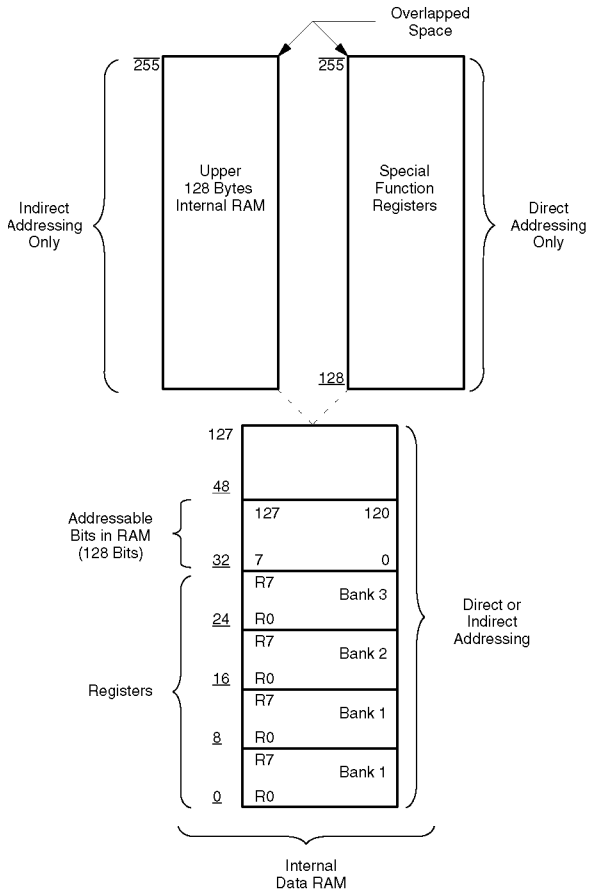

Abbildung 6.3: Darstellung des internen Datenspeichers [1]

register vorgesehen, die aber identisch mit Teilen des unteren Speicherbereiches sind und nur anders angesprochen werden können. Diese Register sind in vier Banken organisiert und können mit Bits im Programm–Status–Byte umgeschaltet werden; ein direkter Zugriff über die absolute Speicheradresse ist aber immer möglich. Der Bitprozessor besitzt als Arbeitsspeicher das Carryflag und kann 256 Bits direkt adressieren.

6.3 Hardware

Alle in der Abbildung 6.1 gezeigten Hardwarekomponenten können nur durch das Setzen und Löschen von speziell dafür vorgesehenen Registern gestartet, gestoppt und kontrolliert werden. Um ein andauerndes Polling der Register zu vermeiden sind fast alle wichtigen Ereignisse interruptfähig, sodaß in insgesamt 15 verschiedenen Interruptroutinen die Bearbeitung vereinfacht wird; zur Unterstützung externer Erweiterungen sind auch zwei externe

Interruptleitungen vorgesehen. Die möglichen Quellen sind in Abbildung 6.4 dargestellt.

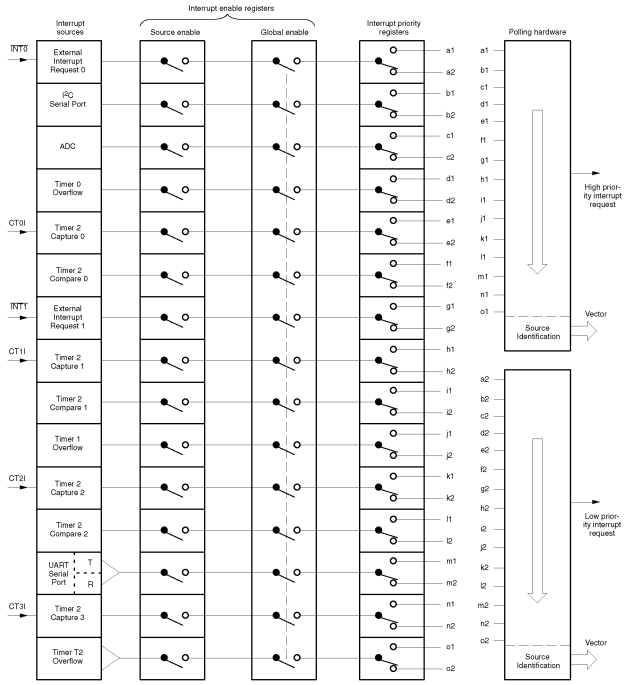

Abbildung 6.4: Darstellung der Interruptverarbeitung [1]

Dort kann man auch die weitere Interruptverarbeitung ersehen. Jeder Interrupt kann einzeln und alle Interrupts können global ein– und ausgeschaltet werden; die Priorität kann auf hoch oder tief gesetzt werden, was bei zeitlich kritischen Vorgängen von Vorteil ist.

Eine weitere Erklärung der verwendeten Hardwarekomponenten ist im Kapitel 9.6 sowie den entsprechenden Kapiteln der Hardware–Erstellung zu finden.

6.4 Befehlsstruktur

Die 8051 Familie hat einen besonders großen Befehlsumfang, sodaß die Programmierung auch komplizierter Berechnungen relativ einfach durchgeführt werden kann. Im folgenden soll eine kleine Übersicht einen Einblick ermöglichen:

arithmetische Befehle: Addition, Subtraktion, Multiplikation, Division, Inkrement und Dekrement

logische Befehle: Und, Oder, exklusives Oder, Komplement, Löschen, Rotieren, Schieben und Nibbletausch

Transfer–Befehle: Move, extern Move, Pop, Push und Tausch

Bit–Befehle: Setzen, Löschen, Tauschen, Und, Oder und Transfer

Programmflußkontroll–Befehle: absoluter und bedingter Sprung, Unterroutinenbedienung und Schleifenbefehle

Kapitel 7

erste Programmüberlegungen

Um überhaupt zu den klaren Definitionen im Kapitel 4 ab Seite 19 zu kommen, mußte ich die Vorgaben genau durchdenken und nachvollziehen. Da zu diesem Zeitpunkt noch keine Prozessorwahl stattgefunden hat[1] und damit auch keinerlei Hardware feststand, wurden vorwiegend die Teile des Programms „vorentwickelt“ die sich auf die Eingabe und die automatische Umschaltung der Arbeitsmodis beziehen. Um aber ein Programm entwickeln zu können, ohne einen Prozessor gewählt zu haben, ist es notwendig, in einem „Pseudocode“ zu schreiben, den man leicht in einen Assemblercode umwandeln kann. Ich habe daher einige wenige Befehle verwendet, die man bei üblichen Prozessoren zumindest mit mehreren Befehlen in einer Makroform darstellen kann.

Ich habe diesen Teil meiner Arbeit deshalb an einer falschen „zeitlichen“ Position eingereiht, da der Pseudocode der Ausgangspunkt für das „echte“ Programm war und so der Zusammenhang besser ersichtlich ist.

In der Tabelle 7.1 sind die wenigen allgemeinen Befehle und Zustände definiert. Das Programm teilt sich in einen Definitionsteil und in das Programm selbst. Ich möchte ausdrücklich darauf aufmerksam machen, daß die Definitionen nur für dieses Pseudoprogramm gedacht sind/waren und keine weitere Bedeutung für das „echte“ Programm haben, denn in den folgenden Kapiteln werden diese ausführlich behandelt. Weiters kann es sein, daß manche Prozeduren nicht in Sinn und Funktion mit denen der verwirklichten Übereinstimmen, da — wie oben erläutert — dieses Programm nur zur Unterstützung der Definitionen und für erste Überlegungen gedacht war. Auch stellt dieses Pseudocodeprogramm nur ein Grundgerüst dar, bei dem sehr viele Funktionen überhaupt nicht oder nur teilweise implementiert sind. Fehler im „echten“ Programm wurden nur dort verbessert, sodaß auch rein logische Fehler in diesem Pseudoprogramm nicht ausgeschlossen werden können. Zuletzt

[1] die Prozessorwahl kann erst nach den Definitionen erfolgen, um alle Anforderungen auch erfüllen zu können

Befehl	*Erklärung*
`TEST`	Testen auf (if then)
`CALL`	Unterroutine aufrufen
`RETURN`	verlassen einer Unterroutine
`GOTO`	Sprung zu einer Marke
`INC`	inkrementiere (+1)
`in`	eingesteckt
`on`	aktiv (gedrückt)

Tabelle 7.1: Erklärung der Pseudobefehle und Zustände

sei noch darauf aufmerksam gemacht, daß das Programm nur wenig dokumentiert ist, da es sich eben nur um einen „ersten schnellen Versuch“ handelt und das Programm leicht verständlich ist.

Ich habe diesen Teil aber trotz aller oben erwähnter Einschränkungen in die Dokumentation übernommen, da es sich um einen der Grundsteine des Projektes handelt und für das prinzipielle Verständnis des Programmablaufes einfacher ist als das endgültige Programm.

7.1 Definitionen

```
DIP-Switches:
0 ... out 3 on/off (falls das 3. Gerät [switch] nicht angeschlossen ist)
1 ... out 2 on/off (falls nur ein normales Gerät angeschlossen ist)
2 ... mouse on/off (Mousemode insgesamt ein/aus)
3 ... scan on/off (schaltet Scan-Mode im Mousemode ein/aus)
4 ... MouseMitte on/off
5 ... progable on/off (schaltet die Ref-Time- und outtime-Programmierung ein/aus)
6 ... Karrenzzeit-Einstellung als Verhältnis der Ref-Time
7 ... --- " ---

Switches:
outtime ... zur Einstellung der outtime-Zeit
Mswitch ... externer Schalter zum Wechseln in den Mousemode
E1 ... externer Bedienungsschalter "Single"
E2 ... externer bedienungsschalter "Double"
E3 ... externer Bedienungsschalter "Select"

Relais-Ports:
out1 ... erstes Gerät
out2 ... zweites Gerät
out3 ... drittes Gerät (switch-Gerät)

Mouse-Ports:
mouse1 ... links
mouse2 ... mitte
mouse3 ... rechts

Errors:
1 ... Anschlußfehler1: nur E3 eingesteckt
2 ... Anschlußfehler2: E1 & E2 eingesteckt und IR off
3 ... Bedienungsfehler1: im Modus 3 E2 länger on als acctime
4 ... Bedienungsfehler2: im Modus 3 E2 beim zweiten mal länger on als acctime

129 ... Fatal1: switchout nicht 1...3 (unmögliches Gerät)
130 ... Fatal2: mousesel nicht 1...4 (unmögliche Bedienungsart)
131 ... Fatal3: Mod nicht 1...5 (unmöglicher Modus, Mod6...7 nur im Mousemod)
```

```
132 ... Fatal4: attention nicht 0...5 (unbekannte Message)
133 ... Fatal5: mouseout nicht 1...4 (unmöglicher Ausgang)

Konstante:
Prell ... Entprellticks
clicktime ... Dauer eines clicks
doubleclicktime ... Zeit zwischen den clicks

Variable:
Mod ... aktueller Modus
Error ... Fehlernummer
ticks ... vergangene Zeitimpulse (zur Zeitmessung)
ticks2 ... vergangene Zeitimpulse (zur relativen Zeitmessung)
wait ... Warteticks
attention ... Messageport der Interruptroutine
outtime ... Länge des einfachen Ausgangsimpulses in ticks
acctime ... Länge der Referenzzeit
Karrtime ... Karrenzzeit im Mod1 zwischen den Umschaltimpulsen (über DIP einstellbar)
switchout ... aktiver Switchausgang ("gerade drann")
mouseout ... aktive Mousetaste ("gerade drann")
mousesel ... aktive Tastenart (click, d-click, ....)
scantime ... Ticks zwischen Scanumschaltungen

attention:
0 ... alles OK
1 ... Moduswechsel (Mod1 ... Mod6)
2 ... Programmieraufforderung
3 ... Hardware-Wechsel Mousemode/Switchmode
4 ... Software-Wechsel von Switchmode in Mousemode
5 ... Software-Wechsel von Mousemode in Switchmode

Modis: (Bit 0...6)
0 ... Modus neu feststellen (z.B. E1 wurde abgesteckt)
1 ... Single (nur E1)
2 ... Single + Select (E1 + E3)
3 ... Double (nur E2)
4 ... Double + Select (E2 + E3)
5 ... IR
6 ... Scanmode Single (nur im Mousemode erlaubt)
7 ... Scanmode Double (nur im Mousemode erlaubt)

Modis: (Bit 7)
on ... Mousemode
off ... Switchmode

mouseout:
1 ... links
2 ... mitte
3 ... rechts

mousesel:
1 ... click
2 ... d-click
3 ... switch
4 ... next
```

7.2 Pseudocode–Programm

In den Kommentaren sind einige Bemerkungen, die für die spätere Realisierung gedacht waren.

`%%Entprell` weist darauf hin, daß an diesen Stellen Entprellzeiten eingefügt werden müssen, wenn an Stelle eines Schließers ein Öffner verwendet wird.

`##Select` zeigt die Möglichkeit einer Vereinfachung an, wenn der Prozessor einen Befehl zur einfachen Mehrfachverzweigung zur Verfügung stellt.

`##Error` wurde zur Kennzeichnung kritischer Stellen verwendet, die bei Änderung der Errornummern auftreten könnten.

```
Zeichenerklärung:
=================
TEST ... Testen auf (if then)
CALL ... Unterroutine aufrufen
RETURN ... verlassen einer Unterroutine
GOTO ... Sprung zu einer Marke
INC ... inkrementiere (+1)

in ... eingesteckt
on ... aktiv (gedrückt)

Programm:
=========
Reset-Routine:
    Setzen der Variablen auf gespeicherte Werte
        time=
        accepttime=
    Aktuelle Tasten / DIP-Stellungen einlesen uns speichern
        DIP(6,7)=on?    ; Einstellung der Karrtime
        E1=in?
        E2=in?
        E3=in?
        Mswitch (mouse-switch)=in?
        IR=on?
    Mod=0
    Interrupts erlauben
    GOTO attention-test:

;***************************** Modus 1 **********************************

Mod1:
    TEST attention=0
    YES
        TEST E1=on
        YES
            ticks=0     ; zur Zeitmessung
            ticks2=0    ; zur Zählung der acctimes
            CALL Entprell
Mod1-1:     TEST ticks<=acctime
            YES
                TEST E1=on
                YES
                    GOTO Mod1-1:
                NO
                    CALL Puls
                    GOTO Mod1:
            NO
                TEST E1=on
                YES
                    TEST attention=1        ; E1 abgesteckt?
                    YES
                        GOTO attention-test:
                    NO

                        TEST ticks2>=acctime
                        YES
                            ticks2=0
                            CALL Nextout
Mod1-2:                     TEST ticks2>=Karrtime
                            YES
                                ticks2=0
                                GOTO Mod1-1:
                            NO
                                TEST E1=on
                                YES
                                    GOTO Mod1-2:
                                NO
                                    GOTO Mod1:
                        NO
                            GOTO Mod1-1:
                NO
                    %%Entprell
                    CALL Nextout
                    GOTO Mod1:
        NO
            GOTO Mod1:
    NO
        GOTO attention-test:

;********************** Modus 2 ***************************************
```

```
Mod2:
    TEST attention=0
    YES
        TEST E1=on
        YES
            ticks=0     ; zur Zeitmessung
            TEST (switchout=3) || (mousemode = on)
            YES
                CALL TOGGLE
                CALL Entprell
Mod2-1:         TEST E1=on      ; Warten auf wieder Loslassen
                YES
                    GOTO Mod2-1:
                NO
                    %%Entprell
                    CALL dontE3 ; Verhindert einen falsch gedrückten E3
                    GOTO Mod2:
            NO
                TEST switchout=2
                YES
                    out2=on
                    GOTO Mod2-3:
                NO
                TEST switchout=1
                YES
                    out1=on
                    GOTO Mod2-3:
                NO
                    Error=129
                    GOTO FatalError:
Mod2-3:     CALL Entprell
Mod2-2:     TEST ticks<=acctime
            YES
                TEST E1=on
                YES
                    GOTO Mod2-2:
                NO
Mod2-5:             TEST ticks<outtime
                    YES
                        GOTO Mod2-5:
                    NO
                        GOTO Mod2-6:
            NO
Mod2-4:         TEST E1=on
                YES
                    GOTO Mod2-4:
                NO
                    GOTO Mod2-6:
        NO
            TEST E3=on
            YES
                CALL stepselect
                GOTO Mod2:
            NO
                GOTO Mod2:
    NO
        GOTO attention-test:

Mod2-6:
    TEST switchout=1
    YES
        out1=off
        %%Entprell
        GOTO Mod2:
    NO
        TEST switchout=2
        YES
            out2=off
            %%Entprell
            GOTO Mod2:
        NO
        Error=129
        GOTO FatalError:


;********************** Modus 3 ****************************************

Mod3:
    TEST attention=0
    YES
        TEST E2=on                          ; zum ersten Mal gedrückt
```

```
        YES
            ticks=0
            CALL Entprell
Mod3-1:     TEST E2=on                      ; Warten auf erstes Loslassen
            YES
                TEST ticks<acctime
                YES
                    GOTO Mod3-1:
                NO
                    Error=3
Mod3-2:             TEST E2=on
                    YES
                        Test attention=1    ; abgesteckter E2!
                        YES
                            Error=0
                            GOTO attention-test:
                        NO
                            GOTO Mod3-2:
                    NO
                        Error=0
                        GOTO Mod3:
            NO
            %%Entprell
Mod3-5:     TEST E2=on                      ; Warten auf zweites Drücken
            YES
                CALL Entprell
Mod3-3:         TEST E2=on                  ; Warten auf zweites Loslassen
                YES                         ; noch nicht
                    TEST ticks<acctime
                    YES
                        GOTO Mod3-3:
                    NO
                        Error=4
Mod3-4:                 TEST E2=on
                        YES
                            Test attention=1    ; abgesteckter E2!
                            YES
                                Error=0
                                GOTO attention-test:
                            NO
                                GOTO Mod3-4:
                        NO
                            Error=0
                            GOTO Mod3:
                NO                          ; losgelassen
                    %%Entprell
                    CALL Nextout            ; 2xgedrückt=Select
                    GOTO Mod3:
            NO                              ; noch kein 2 Drücker
                TEST ticks<acctime
                YES
                    GOTO Mod3-5:
                NO
                    CALL Puls               ; 1xgedrückt
                    GOTO Mod3:

        NO
            GOTO Mod3:
    NO
        GOTO attention-test:

;********************** Modus 4 ****************************************
; entspticht Mod3 mit anderen Aktionen und E3 Behandlung


Mod4:
    TEST attention=0
    YES
        TEST E2=on                          ; zum ersten Mal gedrückt
        YES
            ticks=0
            CALL Entprell
Mod4-1:     TEST E2=on                      ; Warten auf erstes Loslassen
            YES
                TEST ticks<acctime
                YES
                    GOTO Mod4-1:
                NO
                    Error=3
Mod4-2:             TEST E2=on
                    YES
```

```
                        Test attention=1    ; abgesteckter E2!
                        YES
                            Error=0
                            GOTO attention-test:
                        NO
                            GOTO Mod4-2:
                    NO
                        Error=0
                        GOTO Mod4:
            NO
            %%Entprell
Mod4-5:     TEST E2=on                      ; Warten auf zweites Drücken
            YES
                CALL Entprell
Mod4-3:         TEST E2=on                  ; Warten auf zweites Loslassen
                YES                         ; noch nicht
                    TEST ticks<acctime
                    YES
                        GOTO Mod4-3:
                    NO
                        Error=4
Mod4-4:                 TEST E2=on
                        YES
                            Test attention=1    ; abgesteckter E2!
                            YES
                                Error=0
                                GOTO attention-test:
                            NO
                                GOTO Mod4-4:
                        NO
                            Error=0
                            GOTO Mod4:
                NO                          ; losgelassen
                    %%Entprell
                    CALL Puls               ; 2xgedrückt=Puls
                    GOTO Mod4:
            NO                              ; noch kein 2. Drücker
                TEST ticks<acctime
                YES
                    GOTO Mod4-5:
                NO
                    CALL dontE3               ; 1xgedrückt=vergessen
                    GOTO Mod4:

        NO
            TEST E3=on
            YES
                CALL stepselect
                CALL dontE3     ; Warten auf Loslassen von E3
                GOTO Mod4:
            NO
                GOTO Mod4:
    NO
        GOTO attention-test:


;********************** Modus 5 ***************************************
; zu prozessorabhängig (Interruptroutine)

Mod5:

;********************** Modus 6 ***************************************
; Scanmodus Single (nur im Mousemode erlaubt)
; Verhält sich nicht wie Mod2!

Mod6:
    TEST attention=0
    YES
        TEST E1=on
        YES
            CALL Entprell
            CALL Puls
Mod6-1:     TEST E1=on  ; noch immer gedrückt
            YES
                GOTO Mod6-1:
            NO
                %%Entprell
                GOTO Mod6
        NO
            GOTO Mod6:
    NO
```

```
        GOTO attention-test:

;********************** Modus 7 ***************************************
; Scanmodus Double (nur im Mousemode erlaubt)
; Aus Mod4 entwickelt

Mod7:
    TEST attention=0
    YES
        TEST E2=on                          ; zum ersten Mal gedrückt
        YES
            ticks=0
            CALL Entprell
Mod7-1:     TEST E2=on                      ; Warten auf erstes Loslassen
            YES
                TEST ticks<acctime
                YES
                    GOTO Mod7-1:
                NO
                    Error=3
Mod7-2:             TEST E2=on
                    YES
                        Test attention=1    ; abgesteckter E2!
                        YES
                            Error=0
                            GOTO attention-test:
                        NO
                            GOTO Mod7-2:
                    NO
                        Error=0
                        GOTO Mod7:
            NO
            %%Entprell
Mod7-5:     TEST E2=on                      ; Warten auf zweites Drücken
            YES
                CALL Entprell
Mod7-3:         TEST E2=on                  ; Warten auf zweites Loslassen
                YES                         ; noch nicht
                    TEST ticks<acctime
                    YES
                        GOTO Mod7-3:
                    NO
                        Error=4
Mod7-4:                 TEST E2=on
                        YES
                            Test attention=1    ; abgesteckter E2!
                            YES
                                Error=0
                                GOTO attention-test:
                            NO
                                GOTO Mod7-4:
                        NO
                            Error=0
                            GOTO Mod7:
                NO                          ; losgelassen
                    %%Entprell
                    CALL Puls               ; 2xgedrückt=Puls
                    GOTO Mod7:
            NO                              ; noch kein 2. Drücker
                TEST ticks<acctime
                YES
                    GOTO Mod7-5:
                NO
                    GOTO Mod7:              ; 1xgedrückt=vergessen

        NO
                GOTO Mod7:
    NO
        GOTO attention-test:


;********************** dontE3 ****************************************
; testet ob E3 gedrückt ist und fallweise auf Loslassen warten

dontE3:
    TEST E3=on
    YES
dontE3-1:
        TEST E3=on
        YES
            GOTO dontE3-1:
```

```
        NO
            %%Entprell
            RETURN
    NO
        RETURN

;********************** stepselect ************************************

stepselect:
    CALL nextout
step1:
    TEST E3=on
    YES
        GOTO step1:
    NO
        %%Entprell
        RETURN

;********************** Puls ******************************************
; noch keine LEDs bei mouseout


Puls:
    TEST Mod>127        ; Bit 7 gesetzt = Mousemode
    YES
        TEST mousesel=1     ; click
        YES
            CALL clickit
            RETURN
        NO
            TEST mousesel=2     ; d-click
            YES
                CALL clickit
                TEST mouseout=4
                YES
                    RETURN
                NO
                    wait=doubleclicktime
                    CALL Warte
                    CALL clickit
                    RETURN
            NO
                TEST mousesel=3 ;switch
                YES
                    TEST mouseout=4
                    YES
                        CALL clickit
                        RETURN
                    NO
                        TEST mouse[mouseout]=on
                        YES
                            mouse[mouseout]=off
                            RETURN
                        NO
                            mouse[mouseout]=on
                            RETURN
                NO
                    TEST mousesel=4 ; next
                    YES
                        TEST MSwitch=in
                        YES
                            highestmouse=3  ; Variable nur hier benötigt!
                            GOTO Puls-1:
                        NO
                            highestmouse=4  ; Variable nur hier benötigt
                            GOTO Puls-1:
Puls-1:                 INC mouseout
                        TEST mouseout>highestmouse
                        YES
                            mouseout=1  ; letzter out war rechts oder mousemodeoff
                            RETURN
                        NO
                            RETURN
                    NO              ; falsche Bedienungsart
                        Error=130
                        GOTO FatalError:
    NO
        TEST switchout=1    ; Ausgang 1
        YES
            out1=on
            wait=outtime
```

```
            CALL Warte
            out1=off
            RETURN
        NO
            TEST switchout=2    ; Ausgang 2
            YES
                out2=on
                wait=outtime
                CALL Warte
                out2=off
                RETURN
            NO
                TEST switchout=3    ; Ausgang 3
                YES
                    CALL TOGGLE
                    RETURN
                NO                  ; falscher Ausgang
                    Error=129
                    GOTO FatalError

;********************** Clickit ****************************************
; ein Click am entsprechenden Mouseausgang
; (mehrmaliges Aufrufen von Warte wegen Stackstufenbegrenzung)

Clickit:
    TEST MSwitch=not in
    YES
        TEST mouseout=4
        YES
            attention=5
            RETURN
        NO
            GOTO Clickit-1:
    NO
        GOTO Clickit-1:
Clickit-1:
    TEST mouseout=3
    YES
        mouse3=on
        wait=clicktime
        CALL Warte
        mouse3=off
        RETURN
    NO
        TEST mouseout=2
        YES
            mouse2=on
            wait=clicktime
            CALL Warte
            mouse2=off
            RETURN
        NO
            TEST mouseout=1
            YES
                mouse1=on
                wait=clicktime
                CALL Warte
                mouse1=off
                RETURN
            NO
                Error=133
                GOTO FatalError


;********************** Entprell ***************************************
; Wartet die Entprellzeit ab

Entprell:
    wait=Prell
    CALL Warte
    RETURN

;********************** attention-test *********************************
; Wertet die Interruptmessage und die Modusnummer aus und ruft
; entsprechend die Routinen Moduserkennung, Programmierung, Mod1 bis
; Mod7 auf.
; Wenn im SoftwareMousemodus umgesteckt, wird automatisch im Switchmode
; wieder begonnen!

attention-test:
    TEST attention=1        ; ##Select
```

```
    YES
        attention=0         ; Message löschen
        CALL Moduserkennung
att-1:  TEST Mod=0
        YES
            GOTO att-1:
        NO
            Error=0         ; ##Error
            TEST Mwitch=in
            YES
                attention=3     ; zur Sicherheit Mswitch überprüfen
                GOTO att-2:
            NO
att-2:          TEST Mod=1      ; ##Select
                YES
                    GOTO Mod1:
                NO
                    TEST Mod=2
                    YES
                        GOTO Mod2:
                    NO
                        TEST Mod=3
                        YES
                            GOTO Mod3:
                        NO
                            TEST Mod=4
                            YES
                                GOTO Mod4:
                            NO
                                TEST Mod=5
                                YES
                                    GOTO Mod5:
                                NO
                                    Error=131
                                    GOTO FatalError
    NO
        TEST attention=2
        YES
            GOTO Programmierung:
        NO
            TEST attention=3
            YES
                TEST Mswitch=on
                YES
att-3:              TEST DIP3=on
                    YES
                        TEST E1=in
                        YES
                            GOTO Mod6:
                        NO
                            GOTO Mod7:  ; Keine Sicherheitsabfrage !!!!
                    NO
att-4:                  TEST Mod=1      ; ##Select
                        YES
                            Mod=129
                            GOTO Mod1:
                        NO
                            TEST Mod=2
                            YES
                                Mod=130
                                GOTO Mod2:
                            NO
                                TEST Mod=3
                                YES
                                    Mod=131
                                    GOTO Mod3:
                                NO
                                    TEST Mod=4
                                    YES
                                        Mod=132
                                        GOTO Mod4:
                                    NO
                                        TEST Mod=5
                                        YES
                                            Mod=133
                                            GOTO Mod5:
                                        NO
                                            Error=131
                                            GOTO FatalError
                NO
                    GOTO att-2:
```

```
            NO
                TEST attention=4
                YES
                    GOTO att-3:     ; Mousemode=on
                NO
                    TEST attention=5
                    YES
                        GOTO att-2:
                    NO
                        Error=132
                        GOTO FatalError:

;********************** Programmierung *********************************

Programmierung:
; Programmierung von outtime, acctime, scantime

;********************** FatalError *************************************

FatalError:
    out1=off
    out2=off
    out3=off
    mouse1=off
    mouse2=off
    mouse3=off
    GOTO FatalError ; Endlosschleife zur Sicherung

;********************** Nextout ****************************************
; Wählt den nächsten erlaubten Ausgang
; Ein/Ausschalten der LEDS noch nicht implementiert (portabhängig!)

Nextout:
    TEST Mod>127        ; Bit 7 gesetzt = Mousemode
    YES
        CALL Nextmouse
        RETURN
    NO
        TEST switchout=1
        YES
            TEST DIP1=on
            YES
                switchout=2
                RETURN
            NO
                TEST DIP0=on
                YES
                    switchout=3
                    RETURN
                NO
                    RETURN
        NO
            TEST switchout=2
            YES
                TEST DIP0=on
                YES
                    switchout=3
                    RETURN
                NO
                    switchout=1
                    RETURN
            NO
                TEST switchout=3
                YES
                    switchout=1
                    RETURN
                NO
                    Error=129
                    GOTO FatalError:

;********************** Nextmouse **************************************

Nextmouse:
    INC mousesel
    TEST mousesel>4 ; wenn ja, war zuvor next
    YES
        mousesel:=1
        RETURN
    NO
        RETURN
```

```
;********************** Toggle ****************************************
; Toggled den Ausgang 3 wenn Mousemode=off oder (Mousemode=on und
; Mswitch=in)
; ### oder ###
; schaltet in den Mousemode um, indem attention gesetzt wird

Toggle:
    TEST DIP2=on
    YES
        TEST Mswitch=in
        YES
            GOTO Toggle-1:
        NO
            attention=4
            RETURN
    NO
Toggle-1:
        TEST out3=on
        YES
            out3=off
            RETURN
        NO
            out3=on
            RETURN

;********************** Moduserkennung ********************************

Moduserkennung:
    Error=0
    TEST E1=in
    YES
        TEST E2=in
        YES
            Mod=0
            Error=2
            RETURN
        NO
            TEST E3=in
            YES
                Mod=2
                RETURN
            NO
                Mod=1
                RETURN
    NO
        TEST E2=in
        YES
            TEST E3=in
            YES
                Mod=4
                RETURN
            NO
                Mod=3
                RETURN
        NO
            TEST E3=in
            YES
                Mod=0
                Error=1
                RETURN
            NO
                Mod=5
                RETURN

;********************** Warte *****************************************

Warte:
    TEST wait<=0
    YES
        RETURN
    NO
        GOTO Warte:

;********************** Interrupt *************************************
; Noch nicht implementiert, zu prozessorabhängig

Interruptroutine (alle konstante Zeit aufgerufen [Timer-Interrupt])
    Erhöhen von ticks, ticks2 alle x Takte (zur Zeitmessung)
    Wenn Mod>127 und DIP3=on alle scantime CALL Nextmouse
    Erniedrigen von wait alle x Takte (für Wartezähler) und setzten von F0 wenn 0!
    Kontrolle der Tasten E1,E2,E3, Mswitch und der Switches mouse, IR, progable
```

```
    IR,E1,E2,E3 setzt Mod=0 & attention=1
    progable setzt attention=2
    Mswitch setzt attention=3 wenn Mswitch=in
Kontrolle von DIP(6,7) und Einstellung der Karrtime
Ausgabe der Errornummern
```

Kapitel 8

vom Telefonproblem zur multifunktionalen seriellen Schnittstelle

In diesem Kapitel wird die komplette Ansteuerung des Telefons und die sich aus der Problematik ergebende multifunktionale serielle Schnittstelle abgehandelt.

Gleich vorweg muß ich eingestehen, daß ich das „Telefonproblem“ nach eingehenden Rücksprachen mit meinem Betreuer ausgegliedert habe und hier nur eine Teillösung zu finden ist; eine echte Realisierung steht bis dato noch aus und wird (wegen des Umfangs) wahrscheinlich von einem Kollegen übernommen.

8.1 allgemeines zu Schnittstellen

Im Kapitel 4.3 ab Seite 27 wurde bereits einiges über die derzeit (vorallem in Österreich) vorherrschenden Probleme bei der Ansteuerung von GSM–Handys gesagt. Die Hoffnungen, die noch zum Pflichtenheft geführt haben, wurden leider bei der Realisierung nicht erfüllt.

Nach weiteren, insgesamt **vier Monate** langen Forschungen und langwierigen — zum Teil sehr heftigen Diskussionen mit Mitarbeitern der Firma Nokia, wobei ich von verschiedenen Mitarbeitern widersprüchliche bzw. gänzlich falsche Aussagen erhalten habe — stellte sich heraus, daß die direkte Kommunikation mit der Modemkarte des GSM–Handys über eine *RS-232 Schnittstelle* nicht möglich ist, obwohl mir dies zwei Monate zuvor noch schriftlich versichert wurde. Dazu müßte man ein zusätzliches Gerät anschaffen, daß an Stelle des Handy–Akkus eingesetzt wird. Dieses als „Serieller Expander“ bezeichnete Gerät ist damit aber nur in Verbindung mit Nokia–Handys einer Type (z.B. 2110) verwendbar. Dadurch hätte sich eine vollkommene Firmenabhängigkeit und eine Abhängigkeit an ein

bestimmtes Modell ergeben. Dies ist für eine zukunftsorientierte Entwicklung jedoch **nicht tragbar**!

Daher muß nun direkt eine *PCMCIA-Schnittstelle* verwirklicht werden. Dies kann entweder durch ein eigens dafür entwickeltes Gerät oder durch ein fertiges „Fremdprodukt“ geschehen.
Fremdprodukte (wie z.B. der MCDISK von der Firma MPL[1]) haben den Nachteil sehr universell zu sein; dadurch wird die Handhabung komplizierter, die Gehäuse größer und der Preis steigt ebenfalls stark.
Eine Neuentwicklung ist aber im Bereich PCMCIA ein sehr großer Aufwand, wie ich im folgenden Kapitel 8.2 erläutern werde.

Aus all diesen Gründen habe ich mich (in Übereinkunft mit meinem Betreuer) entschieden, die PCMCIA–Ansteuerung in ein eigenes Gerät „auszulagern“, was allein schon aus Platzgründen notwendig ist. Um jedoch die enge Zusammenarbeit mit dem Multiswitch (vorallem in Zusammenarbeit mit der EOG–Brille) nicht zu verlieren, wird eine *multifunktionale serielle Schnittstelle* eingeführt, die das „neue PCMCIA–Schnittstellengerät“ direkt steuern kann. Als weiteres positives Nebenprodukt können auch alle Einstellungen (mehr noch als am Gerät selbst) über einen angeschlossenen PC vorgenommen werden. Dazu wurde ein Verhandlungsprotokoll[2] auf der seriellen Schnittstelle definiert, das eine Verbindung auch mit zukünftigen Erweiterungen ermöglicht.

8.2 PCMCIA

Die PCMCIA–Schnittstelle[3] wurde zunächst in der Version 1.0 für Speicherkarten im mobilen Bereich eingeführt. Der Vorteil gegenüber anderen zu dieser Zeit am Markt befindlichen Produkten lag in der einfachen Handhabung für den Benutzer („Plug and Play“) als auch in der Vielseitigkeit. Über diese Schnittstelle können sowohl dynamische Speicher als auch statische oder gar Festspeicher (EEPROM, Flash–Eprom) angeschlossen werden, sodaß dieser Speicher auch als Diskettenersatz verwendet werden kann.

Bald kam man nicht mehr mit dem Platz auf den scheckkartengroßen Modulen aus und es wurden die Gehäusetypen II und III definiert. Zugleich wollte man auch andere Geräte über diese Schnittstelle an mobile Geräte anbinden können, sodaß die PCMCIA–Versionen 2.0, 2.1 und PCMCIA'95 folgten. Ab diesen Versionen können nun auch andere Speichermedien wie Festplatten aber auch Schnittstellen wie SCSI, Ethernet, und Modemkarten

[1] in Österreich durch die Firma *allmos electronic* in Eisenstadt vertreten

[2] siehe dazu Kapitel 8.3 ab Seite 69

[3] Sehr gut aufbereitete Informationen zu diesem Thema findet man im Buch [11], dem ich auch die Informationen für dieses Kapitel vorwiegend entnommen habe.

verwendet werden. Um jedoch die Kompatibilität mit der Version 1.0 nicht zu verlieren, melden sich alle Karten zunächst als Speicherkarten an. Erst über ein genau definiertes Verhandlungsprotokoll wird die eigentliche Eigenschaft der Karte aktiviert.

Durch die Vielzahl der Anwendungen kam es auch zu einer Vielzahl von benötigten Versorgungsspannungen. Derzeit sind die Werte 3.3, 5 und 12V definiert. Da nicht jede Karte (und schon gar nicht Karten der Version 1.0) mit allen Spannungen zurechtkommt, wurde auch hier ein zum Teil recht kompliziertes Protokoll definiert. Über mechanische Sensoren können Grundtypen erkannt werden (wie z.B. 3.3V Karte), über zwei Pins des Anschlusses kann auf „stromlose“ Weise die genauen „Bedürfnisse“ der Karte geklärt werden. Erst danach darf an den entsprechenden Pins die Spannung zugeschaltet werden; bei Nichtbeachtung dieses Vorganges kann es zur Zerstörung der eingesteckten Karte kommen!

Ein weiteres Problem besteht in der Art der Steckausführung bzw. in der Intension der PCMCIA–Schnittstelle. Jede Karte kann zu jedem beliebigen Zeitpunkt ein– und ausgesteckt werden, ohne daß es zu Problemen kommen darf. Aus diesem Grund ist auch der Softwareaufwand sehr hoch.

Aus allen vorgenannten Gründen zeigt sich schon, daß eine PCMCIA–Schnittstelle eine sehr komplexe Konstruktion ist. Daher wurden eigene Schnittstellentreiber auf den Markt gebracht, die die unterste Schicht im Protokoll übernehmen. Diese ICs haben jedoch aufgrund der hohen Pin–Anzahl der Schnittstelle und des PC–Buses eine enorm hohe Anzahl (ca. 150 bis über 300) an Anschlußpins, was ein entsprechend großes Gehäuse mit sich bringt. Weiters sind diese Schnittstellentreiber alle eigene Microcomputersysteme, um flexibel genug mit den Karten zusammenarbeiten zu können.

Somit besteht die Aufgabe der Entwicklung einer PCMCIA–Schnittstelle für eine Modemkarte in einer kompletten Entwicklung eines eigenen Mircocomputersystems. Aus diesem Grund haben auch einige Hersteller in den Schnittstellen–ICs einen Singlechip–XT integriert, um alle Funktionen in einem Chip zu konzentrieren und so wenig Peripherie wie nötig zu benötigen.

8.3 serielle Schnittstelle

Um die Kommunikation mit der externen Telefonschnittstelle zu gewährleisten, auch wenn sich diese noch in einem frühen Planungsstadium befindet (oder noch garnicht in Angriff genommen wurde), habe ich ein offenes Protokoll mit dem Ziel definiert, viele Geräte mit unterschiedlichen Eigenschaften mit dem Multiswitch zusammenarbeiten lassen zu können. Gleichzeitig besteht damit die Möglichkeit einen PC über eine Standardschnittstelle mit dem Multiswitch zu verbinden, was eine vereinfachte Einstellung der Gerätedaten ermöglicht; weiters können auch damit die Einstellungen auf einem Standard–PC gespei-

chert werden. Spätere Erweiterungen, wie eine Alarmgebung über ein Funkmodul, sind damit ebenfalls einfach zu realisieren und in der aktuellen Version bereits vorgesehen.

Die Anforderungen haben zu einer Entscheidung gegen den I^2C–Bus geführt, der zwar sehr viel flexibler gewesen wäre und auch mehrere Geräte gleichzeitig auf einem Bus zuläßt, aber eben nur von speziellen I^2C–Bustreibern unterstützt wird. Dadurch wäre eine Kommunikation mit einem Standard–PC nicht möglich gewesen. Da für die derzeit geplanten Erweiterungen auch immer nur zwei Geräte miteinander kommuniziern müssen (Multiswitch ↔ Telefonschnittstelle, Multiswitch ↔ PC) ist dieser Nachteil unerheblich.

Somit entschied ich mich für eine RS–232 Schnittstelle ohne Handshaking und einer Baudrate von 9600 und keiner Parität; die Start und Stoppbits sind von der Hardware des Prozessors auf je eines festgelegt. Die Erweiterung der Schnittstelle auf höhere Baudraten oder Parität ist beim 80C562 möglich, jedoch habe ich darauf verzichtet, da andere RS–232 Systeme diese Möglichkeiten vielleicht nicht haben (oder nur mit stark erhöhtem Aufwand) und damit zukünftige Module komplizierter geworden wären. Gleichzeitig sind die Nachteile kaum zu bemerken, da nur sehr wenige Daten ausgetauscht werden, wodurch eine höhere Baudrate nicht notwenig ist.

8.3.1 Definition des Verhandlungsprotokolls

Da an der seriellen Schnittstelle verschiedene Geräte angeschlossen werden sollen, die sowohl Master als auch Slave sein können, muß ein Initialisierungsprotokoll eingeführt werden.

Dabei soll jedes Gerät (Multiswitch als auch externes Gerät) eine Neuverhandlung der Parameter starten können. Dies geschieht durch Senden des Resetzeichens ($FF) und anschließendem Senden der Gerätenummer, die für jedes Gerät eindeutig zugeordnet wird.

Das angesprochene Gerät überprüft, ob es mit dem angegebenen Gerät zusammenarbeiten kann (z.B können zwei Steuersubsysteme nicht miteinander arbeiten, da beide Slaves sind). Ist das der Fall, so wird mit $00 und anschließender eigener Gerätekennung geantwortet; ist das nicht der Fall, wird mit $01 und anschließender eigener Gerätekennung geantwortet.

Ist eine positive Antwort eingetroffen, überprüft das resetauslösende Gerät, ob es mit dem angesprochenen Gerät zusammenarbeiten kann und antwortet selbst positiv oder negativ ($00 oder $01).

Eine graphische Darstellung eines positiven Verhandlungsprotokolls ist in Abbildung 8.1 zu sehen. Das Gerät B muß nicht der Master sein, da auch ein Slave mit dem Verbindungsaufbau beginnen kann.

Sollte es zu einer Kollision zweier Resetanforderungen kommen (z.B. bei gleichzeitig

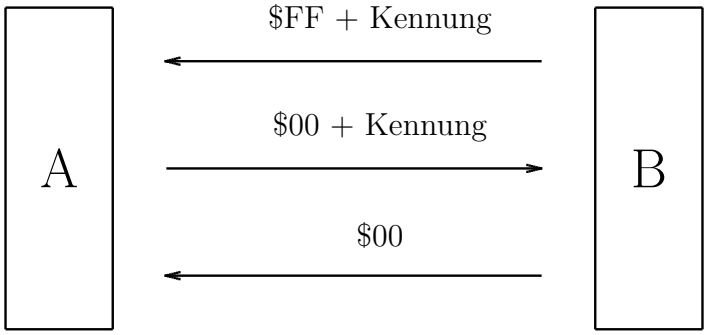

Abbildung 8.1: Darstellung eines „positiven“ Verhandlungsprotokolls

startender Stromversorgung), wartet jedes Gerät die Anzahl der Gerätekennung entsprechenden Nummer von 20ms. Da es keine zwei identischen Gerätekennungen geben kann (identische Geräte zu verbinden macht bei dieser Anwendung keinen Sinn) kommt es beim nächsten Versuch zu keiner Kollision.

Der Multiswitch sendet nur nach einem PowerDown oder einem Reset eine Resetanforderung ab, alle anderen Geräte werden entweder ständig eine Resetanforderung aussenden (bis zu einer erfolgreichen Verbindung), oder gezielt bei Bedarf; Beispiele für dauernd aussendende Geräte sind das Telefonsubsystem, für gezielte Aussendung das PC–Verwaltungsprogramm.

Da der Multiswitch nur eine Resetanforderung aussendet, wird bei einer Kollision keine Wiederholung des Verbindungsaufbaus vom Multiswitch durchgeführt. Dies stellt eine Ausnahme im Protokoll dar, die logischerweise dazu führt, dem Multiswitch die Gerätenummer 0 zuzuordnen.

Kennung	*Gerät*	*Master*	*Slave*
0	Multiswitch	✓	✓
1	Telefonschnittstelle		✓
2	PC	✓	

Tabelle 8.1: derzeit definierte Kennungen der Geräte

Diese Form des Protokolls scheint zwar auf den ersten Blick etwas umständlich für derzeit zwei Subgeräte, ist aber für eine zukunftsorientierte Offenheit notwendig. Die derzeit definierten Kennungen können der Tabelle 8.1 entnommen werden. Nur durch ein Protokoll dieser Art ist sichergestellt, daß weitere Subgeräte entwickelt werden können, die mit allen geeigneten Geräten zusammenarbeiten; neue Versionen erkennen damit z.B.

auch ältere Gerätegenerationen und können entsprechend darauf reagieren. In diesem Fall würde eine neue Version sich zunächst mit der neuen Gerätenummer melden und bei Bedarf (alte Version des Partners) einen erneuten Anmeldeversuch mit einer alten Gerätenummer starten; so kann Abwärtskompatibilität gewährleistet werden.

8.3.2 Zusammenarbeit mit der Telefonschnittstelle

Nach dem erfolgreichen Verbindungsaufbau gibt es drei Möglichkeiten, wie der Multiswitch mit der Telefonschnittstelle kommuniziert. Prinzipiell ist aber immer der Multiswitch der Master, der bis auf den Notruf keine Reaktion von der Telefonschnittstelle (Slave) erwartet und gegebenenfalls ignoriert.

- Solange der Multiswitch nicht im Mod5 oder Mod8 (IR) ist, werden IR-Befehle einfach an die Telefonschnittstelle weitergeleitet; eine Rückmeldung der Telefonschnittstelle wird dabei ignoriert, solange sie nicht zu einer Neuverhandlung des Protokolls führt. Vom Multiswitch wird keine Vorverarbeitung durchgeführt, sodaß die Codes völlig frei gewählt werden können.

- Eine Betätigung des Alarmsensors wird durch das Senden des Alarmcodes (%10101010) signalisiert; der Code wird solange wiederholt, bis eine positive Rückmeldung ($00) erfolgt.

- Im Mod5 und Mod8 (IR) werden viele Codes für die interne Steuerung verwendet. Daher muß der Patient zuerst den Umschaltcode senden ($0F), um in den Telefonmodus zu wechslen. Im Telefonmodus werden wieder alle Codes bis auf den Umschaltcode unbearbeitet an die Telefonschnittstelle weitergeleitet. Im normalen Mod5 oder Mod8 (IR) ohne aktiviertem Telefonmodus werden alle IR–Codes unterdrückt; eine Rückmeldung wird auch hier ignoriert.

8.3.3 Zusammenarbeit mit dem PC

Nach einem positiven Anfangsprotokoll erwartet der Multiswitch (Slave) Befehle vom PC (Master). Derzeit sind nur drei Befehle definiert, die aber eine vollständige Einstellung des Multiswitches erlauben. Die Antwort des Multiswitches hängt dabei vom Befehl ab.

8.3.3.1 Protokoll beim Senden und Empfangen

Prinzipiell wird bei der Kommunikation zwischen Multiswitch und PC die Übertragung und der Erfolg byteweise bestätigt. D.h. zuerst wird vom PC ein Befehl gesendet, der

vom Multiswitch postiv ($00) oder negativ ($01) beantwortet wird. Erst danach darf vom PC das erste Argument übertragen werden, daß ebenfalls vom Multiswitch beantwortet wird; sollte es ein zweites Argument geben, darf auch dieses erst nach positiver Bestätigung des ersten Arguments übertragen werden und wird vom Multiswitch beantwortet. Damit ist sichergestellt, daß der sendende PC eine detaillierte Fehlermeldung an den Benutzer ausgeben kann, da z.B. falsche Befehle von falschen Registern unterschieden werden können.

8.3.3.2 Befehle der Verbindung Multiswitch ↔ PC

Read #

Dieser Befehl erlaubt das Auslesen von Registern. Der Multiswitch gibt als Antwort den Wert des verlangten Registers zurück.

Write # #

Dieser Befehl erlaubt des Beschreiben von Registern. Der Multiswitch gibt als Antwort $01 zurück, wenn das Register nicht existiert oder nicht beschreibbar ist (je nach der Stelle der negativen Antwort) oder $00 wenn alles geklappt hat.

Reset #

Dieser Befehl führt zu einem Neustart mit den aktuellen Werten („Warmstart“) oder den Defaultwerten („Coldstart“) der Register. Auch hier wird byteweise mit positiven oder negativen Zeichen geantwortet. Sollte auch das Argument (Cold/Warm) positiv beantwortet werden, so führt der Multiswitch einen Reset durch und meldet sich mit einem Verbindungsaufbau wieder an.

In der Tabelle 8.2 sind die Befehlsnamen, die Codes und die Argumentsgruppen aufgeführt. Dabei wurden die Codes so gewählt, daß ein gleich großer maximaler Hammingabstand zwischen den Codes besteht, um die Datensicherheit maximal werden zu lassen.

Befehl	*Code*	*Argumentgruppe 1*	*Argumentgruppe 2*
Read	%11000000	α	—
Write	%00011000	α	β
Reset	%00000011	γ	—

Tabelle 8.2: Liste der Befehle der Verbindung Multiswitch ↔ PC

In der Tabelle 8.3 ist die Zuordnungen der Werte zu den Argumentgruppen zu finden. Die Gruppe α gibt das angesprochene Register an, die Gruppe β ist der Datenwert der

Gruppe α								
Code	*Register*	*Write*	*Code*	*Register*	*Write*	*Code*	*Register*	*Write*
1	P1	✓	2	P3	✓	3	P4	✓
4	P5		5	Latch0	✓	6	Latch1	✓
7	Modus	✓	8	mout	✓	9	msel	✓
10	sout	✓	11	atten	✓	12	Error	✓
13	LError	✓	14	Misc	✓	15	Prell	✓
16	ctime	✓	17	dtime	✓	18	otime	✓
19	atime	✓	20	ktime	✓	21	stime	✓
22	ABatt							

Gruppe β	
0	Data
⋮	⋮
255	Data

Gruppe γ	
0	aktuelle Register
255	Defaultwerte

Tabelle 8.3: Liste der Argumentsgruppen und ihre möglichen Werte

gespeichert werden soll und die Gruppe γ gibt an, ob die aktuellen Registerwerte (Warm-Start) oder die Defaultwerte (ColdStart) beim Reset verwendet werden sollen. Zu beachten ist jedoch, daß zwar fast alle Register beschreibbar sind (mit ✓ markiert), alle Register außer den Zeiten jedoch zum Teil von sehr großer Bedeutung für interne Abläufe sind und daher nur mit sehr großer Vorsicht zu verändern sind; Abstürze und unvorhersehbares Verhalten sind durch Manipulationen an den Registern nicht nur möglich sondern zu erwarten!

Daher sollte ein Serviceprogramm für Nichtentwickler drei Ebenen der Bedienung haben. Die *erste Ebene* beinhaltet alle Zeiten, die auch am Gerät einstellbar sind, die *zweite* alle zusätzlichen Zeiten und die *dritte* alle Register. Weiters sollten Programme mit der dritten Ebene nur an besonders geschulte Servicetechniker weitergegeben werden, um Fehlfunktionen, die möglcherweise erst später zu Tage kommen, so weit wie möglich zu verhindern. Es sei eindeutig darauf hingewiesen, daß die dritte von mir vorgeschlagene Ebene nur für die Fehlersuche in Soft– und Hardware gedacht ist und im Normalfall keine sinnvolle Anwendung hat. Daher sollten auch nur die Werte der ersten und zweiten Ebene für eine spätere Verwendung gespeichert werden; alle anderen Register dürfen auf keinen Fall global restauriert werden!

Um die obigen Definitionen der drei Befehle noch deutlicher zu machen, werde ich im folgenden für jeden der Befehle ein Beispiel anführen; alle Beispiele gehen von einem bereits erfolgten postiven Verbindungsaufbau aus:

Angenommen, man will das Register otime auslesen (siehe Abbildung 8.2). In diesem

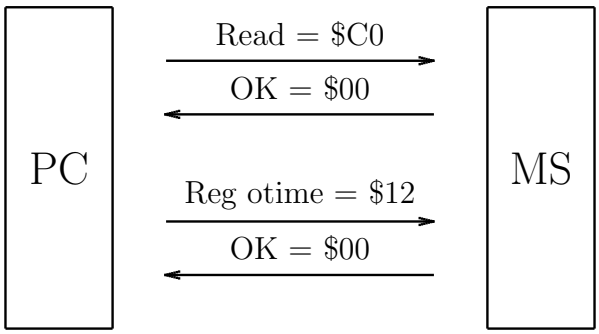

Abbildung 8.2: Beispiel für eine postive Abhandlung eines Readbefehls

Fall muß man also zuerst den Code für den Readbefehl (%11000000 = $C0) an den Multiswitch übermitteln, der postiv ($00) antwortet. Erst danach kann man die Registernummer (in diesem Fall 18 = $12) übermitteln, worauf der Multiswitch mit dem Wert des Registers antwortet.

Sollte man nun das selbe Register mit dem Wert 30 beschreiben wollen (Abbildung 8.3), so geht man ähnlich vor: Zuerst übermittelt man den Code für den Writebefehl (%00011000

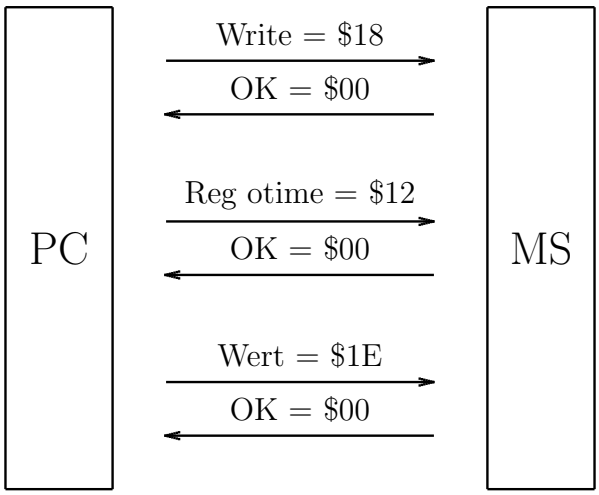

Abbildung 8.3: Beispiel für eine positive Abhandlung eines Writebefehls

= $18) und wartet die positive Antwort ($00) ab. Im zweiten Schritt überträgt man das

gewünschte Register (in diesem Fall wieder 18 = $12), das vom Multiswitch als beschreibbar anerkannt wird. Zuletzt sendet man den zu schreibenden Wert an den Multiswitch (hier 30 = $1E), welcher wieder positiv beantwortet wird.

Das Protokoll für einen Reset (Abbildung 8.2) ähnelt stark dem für den Readbefehl. Will man einen Coldstart durchführen, so sendet man zuerst den Code für den Resetbefehl

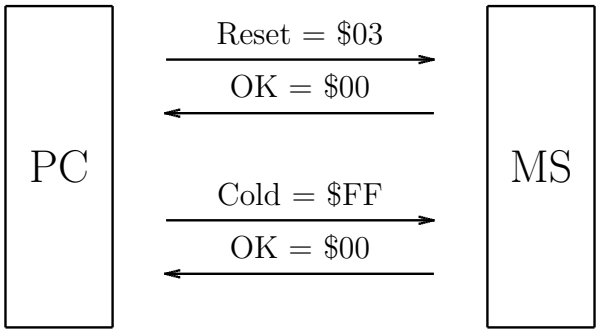

Abbildung 8.4: Beispiel für eine positive Abhandlung eines Coldstarts

(%00000011 = $3), der positiv beantwortet wird; danach überträgt man die Art des Resets (hier Cold = $FF), welche wieder positiv beantwortet wird. Zu beachten ist nun, daß ab dieser Antwort die Verbindung *nicht mehr aufrecht* ist und der Multiswitch nach dem Reset mit einem erneuten Verbindungsaufbau beginnen wird.

Kapitel 9

Software–Entwicklung des Multiswitches

Die Software–Entwicklung war eine der Hauptarbeitsteile des Projektes und wurde *vor* der Hardwareentwicklung durchgeführt, wenn man auch nicht von einer völligen Trennung sprechen kann, da neben der Softwareentwicklung immer die notwendigen Hardwarevoraussetzungen zumindest skizzenhaft geklärt wurden. Die Erstellung der Software habe ich deshalb vor die Hardware–Entwicklung gestellt, da erst mit der Software die letzten Ungereimtheiten beseitigt werden konnten, sodaß die weniger flexiblere Hardware in Angriff genommen werden konnte.

Hier wird die endgültige Version beschrieben und dargestellt. Selbverständlich waren vom Beginn der ersten kompletten Version (Ver 0.70β) bis zur endgültigen Endversion (Ver 1.0) — zumindest dieser Diplomarbeit — viele Fehler zu finden und beheben. Dieser Vorgang wird erst im Kapitel 14 beschrieben, da dieser nicht das Verständnis des Programms behindern soll.

Bei der Beschreibung der Software werden immer ganze Blöcke (Prozeduren, Unterroutinen, oder Gruppen davon) zusammenhängend abgedruckt und beschrieben, wobei nicht einzelne Programmzeilen erklärt werden (dafür ist der Kommentar völlig ausreichend), sondern der dahinterstehende Sinn oder Besonderheiten der Programmierung. Dabei wird aber immer die Reihenfolge im Programm eingehalten, sodaß kein Teil des Programmes fehlt; somit ist im Anhang das Listing nicht ein zweites mal (geschlossen) zu finden! Für eine weitere Verwendung bzw. Weiterentwicklung des Programmes ist ohnehin das Sourcelisting auf Datenträger vorhanden.

Die notwendigen Hardwarekenntnisse über den Prozessor werden im Kapitel 9.6 getrennt von der Software besprochen. Dabei werden aber nur diejenigen Register behandelt, die für dieses Projekt verwendet wurden; eine weitergehende Behandlung ist [9] zu

entnehmen.

9.1 Prinzipielles zur Programmiersprache

Die Software wurde in reinem Assembler–Code geschrieben, was ich bei einem Microcomputer für sinnvoll erachte, obwohl es für die 8051er–Reihe auch Pascal– und C–Compiler gibt. Die Programmierung erscheint zwar auf den ersten Blick etwas komplizierter, dafür hat man aber viel direkteren Zugang zur Hardware und ist nicht auf den Compiler als „Unsicherheitsfaktor“ angewiesen. Da beim Multiswitch–Programm zudem keine höheren Berechnungen vorgenommen werden, wäre der Vorteil einer höheren Programmiersprache noch geringer.

Ich habe für dieses Projekt den Freeware–Assembler der Firma MetaLink in der Version 1.2h verwendet, den man z.B. kostenlos beim FTP–Server der Firma Philips[1] erhalten kann. Dieser Assembler unterstützt zwar keinen Linker und externen Code wie andere vergleichbare Produkte, ist aber relativ schnell, leicht zu handhaben, mit sehr guter Dokumentation versehen und vorallem leicht erhältlich und kostenlos. Die für das Programm notwendigen Besonderheiten bzw. Direktiven werden im Kapitel 9.3 erklärt.

9.2 der Programmierstil

Ich habe während der ganzen Software–Entwicklung immer darauf geachtet, daß zukünftige Erweiterungen leicht einfügbar sind und alle Hardwaredefinitionen über Direktiven angesprochen werden, sodaß eine einfache Anpassung an Hardwareänderungen möglich ist. Die einzigen absoluten Adressen die verwendet wurden sind die vom Prozessor vorgegebenen Reset– und Interruptvektoren bzw. die Grenzen für speziellen Datenspeicher; alle anderen Adressen und Werte wurden über den Assembler dynamisch verwaltet oder über Equates gehandhabt. Sehr selten wurden Konstante (meistens 0) direkt verwendet; diese sind aber entweder aus Prozessorhardwaregründen oder prinzipiellen Programmierstrukturen nicht anders wählbar und deshalb bewußt direkt verwendet worden, um eine versehentliche Änderung zu vermeiden.

9.3 der Assembler

Die folgende kurze Beschreibung dient nur dazu, dem Leser eine kurze Einführung in den verwendeten Assembler zu geben und soll keinesfalls die sehr ausführliche Beschreibung

[1] Internetadresse ftp://ftp.PhilipsMCU.com/pub/bbs/

der Herstellerfirma ersetzen. An dieser Stelle soll nur soviel vermittelt werden, um relativ problemlos den Source–Code lesen zu können.

9.3.1 Allgemeines zum Assembler

Dieser Assembler übernimmt ein mit einem beliebigen Texteditor erstellten Assembler–Source–Code und übersetzt ihn in ein Maschinenprogramm. Dabei wird ein 2–pass–System verwendet. Im ersten Durchgang wird eine Symboltabelle erstellt, die dann beim zweiten Durchgang (bei der „wirklichen“ Übersetzung) verwendet wird; zu diesem Zeitpunkt wird auch das Listing erstellt, in dem die absoluten Adressen, der Maschinencode, der Source–Code und eventuelle Fehlermeldungen beinhaltet sind.

Als Ergebnis des Übersetzungsprozesses wird das Listing im US–ASCII–Format (ohne Tabulatoren) und das Objektfile im Intel–Hex–Format gespeichert.

9.3.2 Symbole

Symbole sind alphanumerische Repräsentanten von Konstanten, Adressen, Makros u.d.gl.; erlaubte Zeichen sind A...Z, a...z, 0...9, ? und _. Zur Unterscheidung von Symbolen und Zahlen darf ein Symbol nicht mit einer Ziffer beginnen; die Groß- und Kleinschreibung wird ignoriert und gilt als gleichwertig. Weiters dürfen Symbole eine Länge von 255 Zeichen haben, wobei aber nur die ersten 32 signifikant sind. Zuletzt sei noch darauf hingewiesen, daß Symbole nicht identisch mit vordefinierten Operatoren (`SHR, NOT, ...`), Registerbezeichnungen (`R0, AB, DPTR, ...`) oder Direktiven (`EQU, DS, ...`) sein dürfen.

9.3.3 Assembler–Kontrollen

Assembler–Kontrollen beginnen mit dem $–Zeichen, müssen alleine in einer Zeile stehen und dienen zur Einstellung bestimmter Funktionen des Assemblers; ein sogenanntes Preferences–File gibt es nicht.

Von mir verwendete Kontrollen sind z.B.:

`$TITLE()`
: In Klammern wird der Titel des Programmes angegeben; damit beginnt im Listing jede neue Seite.

`$MOD...`
: Hiermit wird das entsprechende File mit den vordefinierten Registeradressen geladen.

`$DATE()`
: Das Datum wird auf jede neue Seite im Listing geschrieben.

`$VERSION()`
: Die Versionsbezeichnung wird ebenfalls auf jede neue Seite im Listing geschrieben.

Darüberhinaus gibt es noch viele andere Kontrollen zur Steuerung der Seitenlänge und –Breite im Listing, welche Elemente das Listing enthalten soll und ähnliches.

9.3.4 Sonderzeichen

Einige Zeichen erfüllen im Assembler–Code Sonderfunktionen, die ich hier kurz erläutern möchte.

Nummern–Zeichen (#): Dieses Zeichen kennzeichnet Konstante. Mit diesem Zeichen muß man besonders sorgfältig umgehen, da ein fehlendes Zeichen nicht als Fehler angezeigt wird, sondern der Zahlenwert als absolute Adresse interpretiert wird.

der Punkt (.): Der Punkt an einer Variablen ermöglicht eine Bitadressierung; direkt an den Punkt schreibt man die Bitnummer. Z.B. also P0.5 für das Bit 5 im Port 0.

das Semikolon (;): Das Semikolon leitet einen Kommentar ein. Alle Zeichen nach dem Semikolon werden ignoriert.

das at-Zeichen (@) Dieses Zeichen vor einem Operanden zeigt indirekte Adressierung an, kann aber nur sehr eingeschränkt verwendet werden, da die indirekte Adressierung beim 8051 nur bei sehr wenigen Befehlen möglich ist.

9.3.5 Zahlenbasen

Es sind die Basen Dezimal (default), Octal, Binär und Hexadezimal erlaubt. Zur Unterscheidung von Zahlenbasen werden am Ende von Konstanten Kennbuchstaben verwendet; dies sind d, o, b und h. Weiters müssen Zahlen mit einer Ziffer beginnen. Das heißt, daß z.B. bei hexadezimalen Zahlen bei Bedarf eine 0 voranzustellen ist.

9.3.6 Assembler–Direktiven

Direktiven dienen — einfach ausgedrückt — zur „internen Kommunikation mit dem Assembler“; Speicher kann reserviert, Symbole können definiert werden u.d.gl. In einer Zeile darf immer nur eine Direktive stehen.

Im folgenden einige wichtige Beispiele:

`EQU`

Damit können Konstante Symbolen, aber auch Symbole Symbolen zugeordnet werden; in den Zuordnungen können auch Operatoren und Konstante verwendet werden, sodaß kleinere Berechnungen zum Zeitpunkt der Übersetzung durchgeführt werden können.

`BIT`

Diese Direktive ordnet einem Symbol eine absolute Bitadresse zu; bei der Zuordnung können auch andere Symbole mit dem `BIT`–Operator (.) verwendet werden.

`CODE`

Einem Symbol wird eine absolute Programmadresse zugeordnet.

`DATA`

Hiermit wird einem Symbol eine absolute interne direktadressierbare Datenadresse zugeordnet.

`IDATA`

Wie `DATA` aber für absolute interne indirektadressierbare Datenadresse.

`XDATA`

Wie `DATA` aber für absolute externe Datenadresse.

`CSEG`

Die folgenden Definitionen werden ins Code–Segment geschrieben; mit dem Zusatz `AT` kann gleichzeitig der interne Counter verändert werden.

`BSEG`

Wie `CSEG` aber mit dem Bit–Segment.

`DSEG`

Wie `CSEG` aber mit dem internen direkt adressierbaren Daten–Segment.

`ISEG`

Wie `CSEG` aber mit dem internen indirekt adressierbaren Daten–Segment.

`XSEG`

Wie `CSEG` aber mit dem externen Daten–Segment.

`DS`

Im aktuellen Segment wird die angegebene Anzahl an Bytes reserviert und die Startadresse gegebenenfalls einem Symbol zugeordnet.

`DBIT`

Wie `DS` aber mit Bits.

`DB`

Im Code–Segment werden beliebig viele Datenbytes mit den angegebenen Werten definiert; diese Direktive ist für Tabellen wichtig.

Weitere Direktiven beschäftigen sich unter anderem mit der Segmentcountereinstellung (`ORG, ...`) und konditionierten Übersetzungsteilen (`IF, THEN, ELSE, ...`).

9.3.7 Operatoren

Operatoren werden für Berechnungen zum Zeitpunkt der Übersetzung verwendet. Beispiele dafür sind:
`MOD, SHR, AND, OR, +, -, *, ...`

9.3.8 Mnemonics

Alle im offiziellen Handbuch des 8051 aufgelisteten Mnemonics werden unterstützt[2].

9.4 die Pin–Belegung

Um die Software–Programmierung überhaupt durchführen zu können, mußte eine Pin–Belegung vorliegen. Diese habe ich in Übereinstimmung mit den ersten Skizzen der Hardware wie in Tabelle 9.1 und 9.2 festgelegt In Tabelle 9.1 sind alle Pins des Prozessors nach Ports geordnet aufgelistet. In der zweiten Spalte ist die prozessorspezifische Bitbezeichnung bzw. die elektrische Bezeichnung angegeben, in der dritten ist meine verwendete oder die zweite prozessorspezifische Bezeichnung zu finden; zuletzt ist eine Kurzbeschreibung angegeben. Die Tabelle 9.2 ist ebenso wie die Tabelle 9.1 organisiert, nur daß die beiden Latches eben extern über `MOVX`–Befehle angesprochen werden.

Bei der Definition mußte auf folgende Besonderheiten geachtet werden:

- Port 5 ist nur lesbar
- Port 5 ist nicht bitadressierbar
- Port 5 sollte keine hochfrequenten Signale beinhalten, da sonst Störungen der Analogsignale zu erwarten sind.

[2] siehe Kapitel 9.5 ab Seite 84

Port 0	P0.0	A0/D0	
	P0.1	A1/D1	
	P0.2	A2/D2	
	P0.3	A3/D3	
	P0.4	A4/D4	
	P0.5	A5/D5	
	P0.6	A6/D6	
	P0.7	A7/D7	
Port 1	P1.0	Karr0	DIP
	P1.1	Karr1	
	P1.2	out3on	
	P1.3	out2on	
	P1.4	MoOn	
	P1.5	ScOn	
	P1.6	MoMOn	
	P1.7	PrgOn	
Port 2	P2.0	A8	
	P2.1	A9	
	P2.2	A10	
	P2.3	A11	
	P2.4	A12	
	P2.5	A13	
	P2.6	A14	
	P2.7	A15	

Port 3	P3.0	RXD	Serial in
	P3.1	TXD	Serial out
	P3.2	$\overline{\text{INT0}}$	IR-Ready
	P3.3	$\overline{\text{INT1}}$	Power on/off
	P3.4	Al	Alarm
	P3.5	A3	A3
	P3.6	$\overline{\text{WR}}$	
	P3.7	$\overline{\text{RD}}$	
Port 4	P4.0	E1	E1
	P4.1	$\overline{\text{E1in}}$	E1 in
	P4.2	E2	E2
	P4.3	$\overline{\text{E2in}}$	E2 in
	P4.4	E3	E3
	P4.5	$\overline{\text{E3in}}$	E3 in
	P4.6	A1	A1
	P4.7	A2	A2
Port 5	P5.0	IR0	IR D0
	P5.1	IR1	IR D1
	P5.2	IR2	IR D2
	P5.3	IR3	IR D3
	P5.4	HM	Hardware Mouse
	P5.5	$\overline{\text{HMin}}$	Hardware Mouse in
	P5.6	SBatt	sense Batt.
	P5.7	Pot	Prog Potentiometer

Tabelle 9.1: Belegung der Prozessor–Ports

- die Latches können nur beschrieben und nicht ausgelesen werden
- die Latches können direkt LEDs treiben
- die Latches sind nicht bitadressierbar

Weitere Überlegungen zur Definition der Belegung findet man im Kapitel 9.7.1[3].

[3] ab Seite 97

L0	L0.0	$\overline{\text{MLSel}}$	Mouse L Sel
	L0.1	$\overline{\text{MMSel}}$	Mouse M Sel
	L0.2	$\overline{\text{MRSel}}$	Mouse R Sel
	L0.3	$\overline{\text{MOff}}$	Mouse off
	L0.4	$\overline{\text{MC}}$	Mouse Click
	L0.5	$\overline{\text{MDC}}$	Mouse D–Click
	L0.6	$\overline{\text{MS}}$	Mouse Switch
	L0.7	$\overline{\text{Next}}$	Next

L1	L1.0	$\overline{\text{A1Sel}}$	A1 Sel
	L1.1	$\overline{\text{A2Sel}}$	A2 Sel
	L1.2	$\overline{\text{A3Sel}}$	A3 Sel
	L1.3	$\overline{\text{MOn}}$	Mouse on
	L1.4	MLT	Mouse L Taste
	L1.5	MMT	Mouse M Taste
	L1.6	MRT	Mouse R Taste
	L1.7	$\overline{\text{LBatt}}$	low Batt

Tabelle 9.2: Belegung der externen Latches

9.5 der Befehlssatz

Der Befehlssatz der 8051–Familie ist sehr umfangreich, was — wie schon bei der Prozessorauswahl erwähnt — die Programmierung erleichtert. Im folgenden werde ich die wichtigsten Befehle kurz erläutern, damit das Programm auch für nicht 8051–gewöhnte Personen lesbar ist bzw. manche Programmiereigenheiten klar werden.

9.5.1 Allgemeines

Die Befehlssyntax ist allgemein wie folgt:
`Befehl <Ziel>,<Operand>`
Dabei ist das Ziel vom Befehl durch mindestens 1 Whitespace und das Ziel vom Operanden durch ein Komma zu trennen; zusätzliche Whitespaces sind erlaubt und werden ignoriert.

9.5.2 die wichtigsten Befehle

Im folgenden eine Kurzerklärung der von mir verwendeten Befehle; eine vollständige Beschreibung ist in jedem 8051–Handbuch z.B. der Firmen Siemens, AMD, Philips[4], usw. zu finden.

In der Aufzählung wird immer zuerst eine Kurzbeschreibung aufgeführt, danach die möglichen Ziele und Operanden; dabei sind meistens nicht alle Ziele und Operanden miteinander kombinierbar sind. Bei der Kurzbeschreibung bedeutet `direct` eine Byteadresse im DATA–Bereich, `Rr` ein Register, `@Rr` ein Register zur indirekten Adressierung, `#data` 8–Bit Daten, `#data16` 16–Bit Daten, `bit` eine Bitadresse, `A` den Akkumulator und `C` das Carry–Bit.

[4] in meinem Fall das Userhandbuch des 80C552 [9]

arithmetische Befehle:

`ADD` Zum Ziel wird der Operand hinzuaddiert.
Ziel: `A`
Operand: `#data`, `Rr`, `@Rr`, `direct`

`SUBB` Vom Ziel wird der Operand und das Carry–Bit subtrahiert.
Ziel: `A`
Operand: `#data`, `Rr`, `@Rr`, `direct`

`INC` Das Ziel wird um 1 erhöht.
Ziel: `#data`, `Rr`, `@Rr`, `direct`, `DPTR`

`DEC` Das Ziel wird um 1 erniedrigt.
Ziel: `#data`, `Rr`, `@Rr`, `direct`

logische Befehle:

`ANL` Das Ziel und der Operand werden UND–verknüpft.
Ziel: `A`, `direct`
Operand: `#data`, `Rr`, `@Rr`, `direct`

`ORL` Das Ziel und der Operand werden ODER–verknüpft.
Ziel: `A`, `direct`
Operand: `#data`, `Rr`, `@Rr`, `direct`

`XRL` Das Ziel und der Operand werden EXKLUSIV–ODER–verknüpft.
Ziel: `A`, `direct`
Operand: `#data`, `Rr`, `@Rr`, `direct`

`CLR` Das Ziel wird auf 0 zurückgesetzt.
Ziel: `A`

`CPL` Das Komplement wird errechnet.
Ziel: `A`

`RL` Das Ziel wird um 1 Bit nach links rotiert.
Ziel: `A`

`RR` Das Ziel wird um 1 Bit nach rechts rotiert.
Ziel: `A`

Transfer–Befehle:

`MOV` Der Operand wird ins Ziel kopiert.
Ziel: `A`, `Rr`, `@Rr`, `direct`, `DPTR`
Operand: `#data`, `Rr`, `@Rr`, `direct`, `#data16`

`MOVC` Ein Byte aus dem Code–Bereich wird ins Ziel kopiert.
Ziel: `A`
Operand: registerindirekt mit `A` und `DPTR` oder `PC` als Basisregister
Hinweis: zur Tabellenbearbeitung notwendig

`MOVX` Ein Byte wird vom Operanden ins Ziel kopiert, wobei einer der beiden im externen Speicherbereich liegt.
Ziel: `A`, `@Rr`, `@DPTR`
Operand: `A`, `@Rr`, `@DPTR`

`PUSH` Ein Byte wird auf den Stack kopiert
Ziel: `direct`

`POP` Ein Byte wird vom Stack ins Ziel kopiert
Ziel: `direct`

Bit–Befehle:

`CLR` Löscht das Zielbit
Ziel: `bit`, `C`

`SETB` Setzt das Zielbit
Ziel: `bit`, `C`

`CPL` Errechnet das Komplement des Ziels
Ziel: `bit`, `C`

`MOV` Kopiert den Operanden ins Ziel
Ziel: `bit`, `C`
Operand: `bit`, `C`
Hinweis: `MOV bit,bit`ist nicht möglich!

Programmfluß–Kontroll–Befehle:

`ACALL` Unterroutinensprung zu einer 11–Bit–Adresse

`LCALL` Unterroutinensprung zu einer 16–Bit–Adresse

`RET` Rückkehr aus einer Unterroutine

`RETI` Rückkehr aus einer Interruptroutine

`AJMP` Sprung zu einer 11–Bit–Adresse

`LJMP` Sprung zu einer 16–Bit–Adresse

`SJMP` Sprung zu einer relativen 8–Bit–Adresse

`JZ` Sprung zu einer relativen 8–Bit–Adresse wenn `A` gleich 0 ist.

`JNZ` Wie `JZ`, aber bei ungleich 0

`JC` Sprung zu einer relativen 8–Bit–Adresse wenn `C` gesetzt.

`JNC` Wie `JZ`, aber `C` nicht gesetzt.

`JB` Sprung zu einer relativen 8–Bit–Adresse wenn Operand gesetzt.
Operand: `bit`

`JNB` Wie `JB`, aber bei Operand nicht gesetzt.

`JBC` Wie `JB`, aber zusetzlich wird der Operand gelöscht.

`CJNE` Vergleicht Ziel und Operand und springt zu einer relativen 8–Bit–Adresse, wenn die beiden ungleich sind; zum anschließenden Größenvergleich wird das `C`–Bit entsprechend gesetzt.
Ziel: `A`, `Rr`, `@Rr`
Operand: `direct`, `#data`

`DJNZ` Das Ziel wird dekrementiert und die relative 8–Bit–Adresse wird angesprungen wenn das Ergebnis ungleich 0 ist.
Ziel: `direct`, `Rr`

9.6 Prozessor–Hardware

Im folgenden werden die für dieses Projekt relevanten Hardwarekomponenten besprochen und die zugehörigen Register erläutert. Die Reihenfolge habe ich dabei gegenüber dem Handbuch [9] nicht verändert, um ein leichteres Nachschlagen zu ermöglichen.

9.6.1 Ports

Die Ports können i.a. frei als Eingänge und Ausgänge bitweise eingesetzt werden. Einschränkungen ergeben sich jedoch durch die Doppelnutzung der Pins mit anderen Hardwarekomponenten (Adressbus, Interrupts, Timer, ADC, ...). Angesprochen werden die Ports über interne Spezialregister (P0...P5), die bei Leseoperationen dem Eingangsregister und bei Schreiboperationen dem Ausgangsregister entsprechen. Alle Spezialregister außer P5 sind bitadressierbar, was eine enorme Erleichterung für den Programmierer mit sich bringt.

Grundsätzlich können die Portbuffer in vier Gruppen unterteilt werden:

1. Standardbuffer
2. Adressbuffer
3. I^2C–Buffer
4. Analogbuffer

9.6.1.1 Standardbuffer

Die Ports 2, 3, 4 und die Portbits 1.0 bis 1.5 sind Standardbuffer. Diese haben einen starken Transistor gegen GND, aber eine komplizierte Anordnung gegen VCC (siehe Abbildung 9.1). Der „strong–pullup“ ist nur im Augenblick des Umschaltens aktiv, um eine steilere

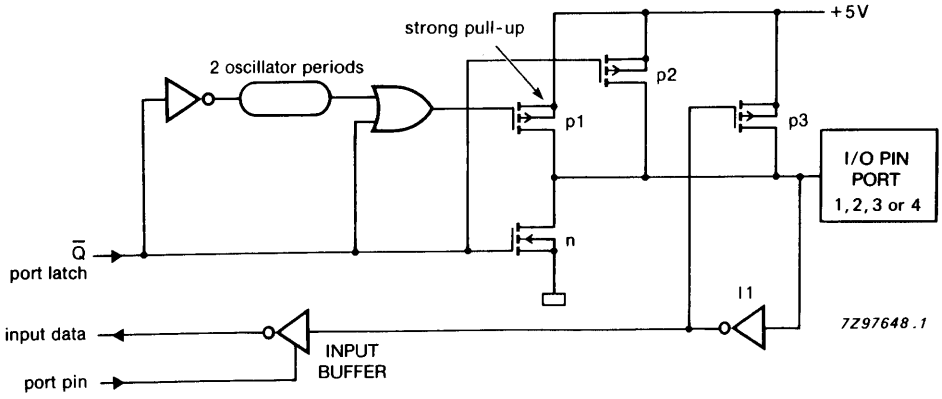

Abbildung 9.1: Aufbau eines Standardbuffers [9]

Übergangsflanke zu erzielen. Im „Normalbetrieb“ wird das Portbit nur durch den schwachen Transistor p3 mit VCC verbunden, der hier wie ein Pullup–Widerstand im Bereich 100kΩ wirkt. Diese Anordnung wurde verwirklicht, weil somit der gleiche Port als Ausgang wie auch als Eingang verwendet werden kann. Um den Port als Eingang zu schalten, muß man nur den Port auf 1 setzen; externe Quellen können dann den Eingang beliebig nach GND ziehen.

Vorsicht ist bei dieser Anordnung aber bei zwei Details geboten:
Bei modifizierenden Befehlen (Read–Modify–Write) wird beim Lesen immer das Ausgangslatch und nicht der aktuelle Wert der Portpins eingelesen. Somit können unerwartete Veränderungen auftreten.
Zweitens verhält sich das Port bei einem Powerdown derart, daß der starke Pull–up–Transistor andauernd eingeschaltet wird, wenn das Ausgangslatch mit einer 1 geladen ist, was zu einem hohen Stromverbrauch führt. Daher müssen vor einem Powerdown alle Ports auf 0 gesetzt werden (auch Port 2, das ansonsten als Adressbus verwendet wird!).

9.6.1.2 Adressbuffer

Port 0 besteht ausschließlich aus Adressbuffer. Dies sind Portbits, die sowohl nach VCC als auch GND einen „starken“ Transistor (strong pullup and pulldown) besitzen. Wenn der Prozessor in den Power–Down–Mode geht, werden beide Transistoren deaktiviert, sodaß die Adressbuffer „floaten“. Bei Verwendung als Port ist der obere Transistor immer deaktiviert, sodaß Open–Drain–Ausgänge entstehen.

9.6.1.3 I^2C–Buffer

Diese Buffer werden nur an den beiden I^2C–Leitungen im Port 1 verwendet und sind Open–Drain–Ausgänge. Sollte man daher diese Bits als Eingänge verwenden wollen, so muß man externe Pull–up–Widerstände verwenden.

9.6.1.4 Analog–Buffer

Diese werden im Port 5 verwendet und sind keine eigentlichen Buffer, da diese Portbits eine Kombination von Analogeingang und Digitaleingang sind und keinen Ausgangstransistor besitzen.

9.6.2 ADC — Analog–Digital–Konverter

Der ADC ist ein Successive-Approximation–System, mit einer Umsetzzeit von 50 Prozessorzyklen. Zur Verwendung des ADCs stehen zwei Register zur Verfügung; mit `ADCON` kann die Steuerung übernommen werden und in `ADCH` steht der digitalisierte Wert; `ADCON` ist **nicht** bitadressierbar. Eine Konversion läuft im allgemeinen wie folgt ab: Man wählt über die `AADR?`–Bits den gewünschten Ausgang aus, setzt das `ADCI`–Bit mit einem `ORL`–Befehl (keine Bitadressierung!) und wartet bis das `ADCI`–Bit gesetzt ist. Dann wird das Bit gelöscht, um weitere Konversionen zu ermöglichen, und kann den Datenwert aus dem `ADCH`–Register retten.

9.6.3 Timer 0 und 1

Beide Timer sind im Aufbau und der Verwendung nahezu identisch. Der einzige für dieses Projekt relevante Unterschied besteht darin, daß nur der Timer 1 für die Bitratengeneration der seriellen Schnittstelle verwendbar ist. Ich werde daher im folgenden keinen Unterschied zwischen den beiden Timern in der Beschreibung machen; die Register werden anstelle der Nummer (0 oder 1) mit einem Fragezeichen an der betreffenden Stelle angegeben.

	7	6	5	4	3	2	1	0
ADCON	ADC.1	ADC.0	ADEX	ADCI	ADCS	AADR2	AADR1	AADR0

Bit	*Erklärung*
ADC.1	Bit 1 des ADC–Ergebnisses (80C552)
ADC.0	Bit 0 des ADC–Ergebnisses (80C552)
ADEX	Erlaubt externen Hardware–Start des ADC
ADCI	ADC–Interrupt–Flag: Wird gesetzt, wenn die Konversion beendet ist und der Wert in `ADCH` gültig ist. Eine neue Konversion kann nur gestartet werden, wenn das Bit gelöscht ist.
ADCS	Durch Setzen dieses Bits wird die Konversion gestartet; bleibt gesetzt, bis die Konversion beendet ist.
AADR2	höchstes Bit der „Adresse“ zur Einstellung des Eingangs. Mit diesem und den nachfolgenden zwei Bits, können die 8 Eingänge des Ports 5 ausgewählt werden.
AADR1	siehe oben
AADR0	siehe oben

Tabelle 9.3: das `ADCON`–Register

Jeder Timer kann in vier verschiedenen Moden betrieben werden, wobei ich jedoch nur 2 verwendet habe und hier erläutern werde.

9.6.3.1 Mode 1

In diesem Modus werden alle 16 Bit des Timers verwendet. Das obere Byte kann mit dem Register `TH?` und das untere Byte mit dem Register `TL?` angesprochen werden. Bei jedem Überlauf des Timers wird das Interruptbit `TF?` gesetzt, sodaß man periodische Interruptaufrufe mit beliebigen Intervallen in der 16–Bit–Grenze produzieren kann, da die 16–Bit des Timers jederzeit per Software veränderbar sind. Weiters können beide Timer als Counter für externe Prozesse verwendet werden, was hier aber nicht weiter erläutert werden soll.

9.6.3.2 Mode 2

Der Modus 2 ist in den meisten Hinsichten identisch mit Mode 1. Statt 16 Bit werden jedoch nur die unteren 8 Bit als Timer verwendet. Gleichzeitig werden die nun unge-

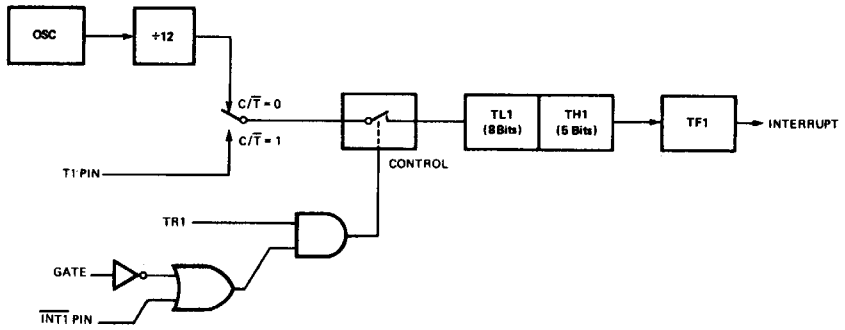

Abbildung 9.2: Darstellung zum Mode 1 der Timer 0 und 1 [9]

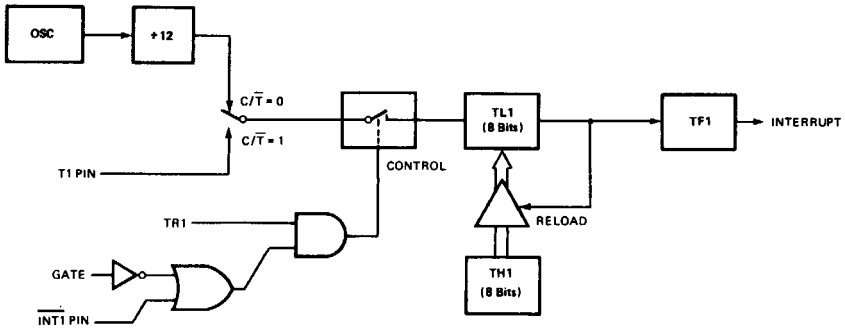

Abbildung 9.3: Darstellung zum Mode 2 der Timer 0 und 1 [9]

nutzten oberen 8 Bit des Counters automatisch bei jedem Überlauf in die unteren 8 Bit kopiert, sodaß ohne weiteren Softwareaufwand und weitere Interruptaufrufen jede beliebige Intervallzeit im 8–Bit–Rahmen erzeugt werden kann.

9.6.3.3 Spezialregister der Timer 0 und 1

Es gibt zwei Spezialregister, die sich die beiden Timer untereinander aber auch mit anderen damit verbundenen Funktionen teilen.

Das `TMOD`–Register ist in das obere und das untere Nibble nach Timer geteilt; die oberen 4 Bit gehören zum Timer 1, die unteren zum Timer 0. Die Funktionen der einzelnen Steuerbits sind einfach und sind in der Tabelle 9.4 erläutert. Alle Bits werden bei einem Reset gelöscht.

Das `TCON`–Register steuert zum einen die Timer direkt mit den Run–Bits (`TR?`) und den Interruptbits (`TF?`), zum anderen die externen Interrupts (`IE?, IT?`); die genaue Beschreibung ist der Tabelle 9.5 zu entnehmen. Alle Bits werden bei einem Reset gelöscht.

	7	6	5	4	3	2	1	0
TMOD	Gate	C/$\overline{\text{T}}$	M1	M0	Gate	C/$\overline{T}$	M1	M0

Bit	*Erklärung*
Gate	Damit kann die Laufkontrolle entweder mittels Software (TR?–Bit im TCON–Register) oder mit dem externen Interrupteingängen gesteuert werden.
C/$\overline{\text{T}}$	schaltet zwischen Counter und Timerfunktion um
M1	Diese zwei Bits dienen der Einstellung der vier Moden
M0	siehe oben

Tabelle 9.4: TMOD–Register

	7	6	5	4	3	2	1	0
TCON	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Bit	*Erklärung*
TF1	Timer 1 Interruptflag: Dieses Bit wird bei einem Überlauf des Timers 1 gesetzt und automatisch beim Beginn der Interruptserviceroutine gelöscht.
TR1	Wenn dieses Bit gesetzt ist, beginnt Timer 1 zu laufen.
TF0	siehe TF1 für Timer 0
TR0	siehe TR1 für Timer 0
IE1	Dieses Bit wird gesetzt, wenn der externe Interrupteingang ausgelöst wird; das Bit wird automatisch beim Beginn der Interruptserviceroutine gelöscht.
IT1	Wenn dieses Bit gesetzt ist, reagiert der externe Interrupt 1 auf Flanken, sonst auf Pegel.
IE0	siehe IE1 für Eingang 0
IT0	siehe IT1 für Eingang 0

Tabelle 9.5: TCON–Register

9.6.4 serielle Schnittstelle

Die allgemeine serielle Schnittstelle — und nur die soll hier besprochen werden — ist ein full–duplex Port; d.h. Senden und Empfangen kann zur selben Zeit geschehen. Die

Schnittstelle kann in vier Moden betrieben werden, wobei aber nur einer der Spezifikation aus dem Kapitel 8.3 entspricht; dieser Modus 1 wird hier auch als einziger beschrieben.

Die Softwareansteuerung des seriellen Ports ist denkbar einfach. Nach der Initialisierung der entsprechenden Spezialregister (siehe Kapitel 9.6.4.2) kann ein Byte versendet werden, indem man es in das `SOBUF`–Register schreibt; der Empfang kann durch Lesen vom `SOBUF`–Register realisert werden. Um das Schreiben mehrerer Bytes oder den Empfang von Daten zeitlich koordinieren zu können, wird ein Interrupt bei jedem vollendet gesendeten Byte oder jedem vollständig empfangenen Byte ausgelöst. Da beide Ereignisse den gleichen Interrupt auslösen, muß die Serviceroutine selbst anhand der Interruptbits die beiden Auslöseursachen unterscheiden. Die interruptauslösenden Bits werden daher nicht von der Hardware automatisch gelöscht; dies muß die Software übernehmen, da ansonsten die Interruptserviceroutine sofort nach dem Verlassen wieder durch das „gleiche“ Bit aufgerufen wird.

Da das Empfangsschieberegister und das Leseregister verschieden sind, kann ein weiteres Byte empfangen werden, während man das letzte bearbeitet. Hat man nach Vollendung des zweiten Bytes das erste noch immer nicht abgeholt, geht es verloren.

9.6.4.1 Baudratengeneration

Die Baudratengeneration geschieht durch den Timer 1. Am einfachsten ist sicher der Timer im Modus 2 (autoreload), wofür auch eine Tabelle für die gängigsten Baudraten im Userhandbuch [9] zu finden ist bzw. ein Auszug daraus in Tabelle 9.6. Daher braucht

Rate	`SMOD`	`TH1`
19200	1	FDh
9600	0	FDh
4800	0	FAh
2400	0	F4h
1200	0	E8h

Tabelle 9.6: Tabelle zur Einstellung des Timers 1 im Modus 2 bei einer Quarzfrequenz von 11.059MHz

dieses Thema nicht mehr weiter behandelt zu werden.

9.6.4.2 Spezialregister der seriellen Schnittstelle

Die serielle Schnittstelle 0 hat nur ein Spezialregister, das fast alle Funktionen in sich vereinigt. Zu beachten ist vorallem das Bit `REN`, das zuerst gesetzt werden muß, bevor

	7	6	5	4	3	2	1	0
SOCON	SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Bit	*Erklärung*
SM0	Bit zur Einstellung des seriellen Moduses
SM1	siehe oben
SM2	Multiprozessorkommunikation im Mode 2 und 3
REN	Schaltet den seriellen Empfang ein
TB8	9. Datenbit beim Senden im Mode 2 und 3
RB8	9. Datenbit beim Senden im Mode 2 und 3
TI	Interruptbit wird gesetzt, wenn das aktuelle Byte gesendet ist
RI	Interruptbit wird gesetzt, wenn ein Byte vollständig empfangen wurde

Tabelle 9.7: SOCON–Register

man Daten empfangen kann.

Das Bit SMOD zur Kontrolle der Baudratengeneration ist im PCON–Register (siehe Kapitel 9.10) untergebracht und dient der Verdopplung der Baudrate; für die üblichen Datenraten wird dieses Bit jedoch kaum benötigt.

9.6.5 Interrupts

Die prinzipielle Vorgehensweise der Interruptverarbeitung wurde bereits im Kapitel 6.3 in der Abbildung 6.4[5] dargestellt. Die Steuerung der Interrupts erfolgt durch vier Register, die im folgenden erläutert werden sollen; die Einstellungen der einzelnen Hardwarekomponenten zur Interruptauslösung ist in den entsprechenden spezifischen Kapiteln (wenn nötig) beschrieben.

9.6.5.1 Interruptenable–Register

Mit den beiden Interruptenableregistern können alle Interrupts gleichzeitig (EA–Bit) oder jeder Interrupt einzeln ein– und ausgeschaltet werden. Dazu muß man nur das entsprechende Bit in den Registern IEN0 und IEN1 setzen (=ein) oder löschen (=aus). Eine Aufstellung der einzelnen Bits ist in der Tabelle 9.8 zu finden; das IEN1–Register wird

[5] auf Seite 51

	7	6	5	4	3	2	1	0
IEN0	EA	EAD	ES1	ES0	ET1	EX1	ET0	EX0

Bit	*Erklärung*
EA	alle Interrupts
EAD	ADC
ES1	I^2C
ES0	serielle Schnittstelle
ET1	Timer 1
EX1	externer Interrupt 1
ET0	Timer 0
EX0	externer Interrupt 0

Tabelle 9.8: IEN0–Register

nicht erläutert, da nur verschiedene Interrupts des Timers 2 enthalten sind.

9.6.5.2 Interruptpriority–Register

Jedem Interrupt kann entweder eine hohe oder eine niedere Priorität zugeordnet werden. Dies geschieht durch Setzen (=hoch) oder Löschen (=niedrig) der entsprechenden Bits;

	7	6	5	4	3	2	1	0
IP0	—	PAD	PS1	PS0	PT1	PX1	PT0	PX0

Die Anordnung ist die selbe wie im IEN0–Register.

Tabelle 9.9: IP0–Register

eine Aufstellung der Bits ist der Tabelle 9.9 zu finden, IP1 enthält wieder nur Interrupts des Timers 2.

9.6.6 Power–Reduction–Modes

Der 80C552 kennt zwei Modis, die den Stromverbrauch senken: den Idle–Mode und den Power–Down–Mode.

9.6.6.1 Idle–Mode

Der Idle–Mode hält die CPU und manche Hardware an, sodaß sich der Stromverbrauch auf ca 10mA einschränkt; dieser Modus kann durch externe oder interne Interrupts wieder verlassen werden.

9.6.6.2 Power–Down–Mode

Der Power–Down–Mode schaltet alle Hardware aus (auch den Resonator!), sodaß sich der Stromverbrauch auf maximal 100μA absenkt; dieser Modus kann jedoch nur noch durch einen Hardwarereset beendet werden.

Zu beachten ist bei beiden Moden, daß zu den angegebenen Stromverbrauchswerten der Strom der Ports hinzukommt, sodaß — wie im Kapitel 9.6.1 beschrieben — auch im Power–Down–Mode ein Verbrauch von einigen 10 Milliampere vorkommen kann.

9.6.6.3 Spezialregister der Power–Reduction-Modes

Beide Modis werden durch setzen eines Bits im `PCON`–Register (siehe Tabelle 9.10) gestartet. Sollten beide Bits gleichzeitig gesetzt werden, so wird der Power–Down–Modus gestartet.

	7	6	5	4	3	2	1	0
`PCON`	SMOD	—	—	WLE	GF1	GF0	PD	IDL

Bit	*Erklärung*
SMOD	verdoppelt die Baudrate (siehe Kapitel 9.6.4.2)
WLE	Dieses Bit muß gesetzt werden um den Watchdogtimer (Timer 3) laden zu können.
GF1	frei nutzbares Userbit
GF0	frei nutzbares Userbit
PD	Power–Down–Bit
IDL	Idle–Bit

Tabelle 9.10: `PCON`–Register

9.7 das Programm „TheProg“

Das Programm des Multiswitches wird nun in zwei Teilen aufgelistet und Abschnittsweise erläutert. Dabei wird immer die Zeilenreihenfolge eingehalten, um insgesamt ein komplet-

tes Programm aufzuführen. Für die Definitionen werden noch einige Erläuterungen zum internen Speichersystem des Prozessors gegeben.

9.7.1 die Definitionen

Für die Definitionen der Speicherbereiche ist es enorm wichtig, die genauen Speichergrenzen zu kennen, damit man später bei der Programmierung so einfach wie nur möglich auf die verschiedenen Register und deren Bits zugreifen kann. Manche Befehle (wie z.B. `cjne` können) nur mit Registern als Argumenten arbeiten. Daher ist es günstig – wie ich später zeigen werde — manche Variable auf Speicherbereiche zu legen, die gleichzeit als Register angesprochen werden können.

Daher soll die Abbildung 9.4 die Grenzen und Adressierungsmöglichkeiten klarstellen. Auf der linken Seite ist der untere Speicherbereich dargestellt, der sowohl direkt als auch indirekt adressiert werden kann. Weiters sind die direkten Adressen der vier Registerbanken eingezeichnet. Auf der rechten Seite sind die Spezialregister mit ihren Bitadressen angegeben; zu beachten ist, daß die Spezialregister nur direkt adressiert werden können (siehe Kapitel 6.2).

9.7.1.1 der Kopf

```
$TITLE(TheProg)
$VERSION($VER 1.00)
$DATE(April 97)
$NODEBUG
$NOPAGING
$MOD552
```

Hiermit wird der Titel, die Version und das Datum des Programmes gesetzt. Weiters wird verhindert, das Debugcode erzeugt wird und das das Listing in Seiten unterteilt wird. Zuletzt werden die Sonderdefinitionen des 80C552 Prozessors geladen.

9.7.1.2 Stack

```
 ; Stack
 ; =====

        ISEG AT 128
Stack:  DS 128
```

Der Stack wird im indirekt adressierbaren („oberen“) Speicherbereich mit einer Größe von 128 Byte festgelegt.

9.7.1.3 Register

```
 ; Registerbanken
 ; ==============

        DSEG AT 0
```

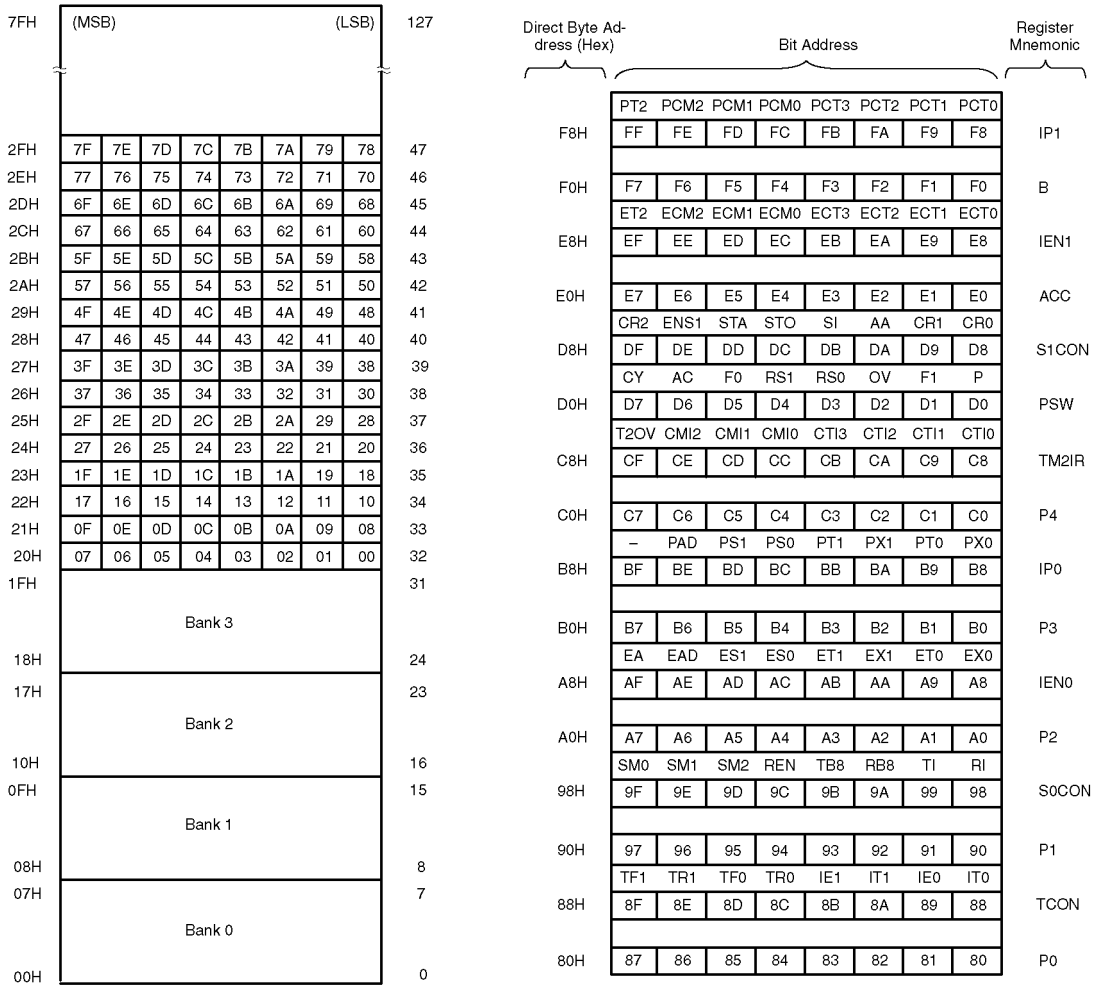

Abbildung 9.4: Darstellungen des RAMs des 80C552: links unterer Bereich mit Registerbanken [1] — rechts Spezialregister mit Bitadressen [1]

```
; Bank 0

R0_0:   DS 1            ; für Unterroutinen
R1_0:   DS 1            ; für Unterroutinen
R2_0:   DS 1            ; frei
Modus:  DS 1            ; aktueller Modus
Modr    EQU R3          ; zur Registeradressierung
mout:   DS 1            ; aktive Mousetaste ("gerade drann")
moutr   EQU R4          ; zur Registeradressierung
msel:   DS 1            ; aktive Tastenart (click, d-click, ...)
mselr   EQU R5          ; zur Registeradressierung
sout:   DS 1            ; aktiver Switchausgang ("gerade drann")
soutr   EQU R6          ; zur Registeradressierung
atten:  DS 1            ; Messageport
att     EQU R7          ; zur Registeradressierung

; Bank 1                ; in Verwendung für Interrupts Stufe 0

R0_1:   DS 1
R1_1:   DS 1
R2_1:   DS 1
```

```
R3_1:   DS 1
R4_1:   DS 1
R5_1:   DS 1
R6_1:   DS 1
R7_1:   DS 1

; Bank 2                ; in Verwendung für Interrupts der Stufe 1

R0_2:   DS 1
R1_2:   DS 1
R2_2:   DS 1
R3_2:   DS 1
R4_2:   DS 1
R5_2:   DS 1
R6_2:   DS 1
R7_2:   DS 1

; Bank 3

R0_3:   DS 1            ; komplett frei
R1_3:   DS 1
R2_3:   DS 1
R3_3:   DS 1
R4_3:   DS 1
R5_3:   DS 1
R6_3:   DS 1
R7_3:   DS 1
```

Wie in der Einleitung erläutert, habe ich die Registeradressen für die häufigsten Variblen genutzt, um bei Abfragen mit `cjne` diese über Register ansprechen zu können. Besonderen Augenmerk muß man dabei jedoch bei der Programmierung auf die aktuelle Registerbank legen. Um eine einfachere Zuordnung der Variablennamen zu den Registernamen zu ermöglichen, habe ich mittels `EQU` Pseudonamen für die Register vergeben.

9.7.1.4 Byteadressen mit Bitadressierung

```
 ; Byte Direktadressen (mit Bitadressen)
 ; ======================================

        DSEG AT 32

Latch0: DS 1            ; Ausgabelatch 0
MLSel   BIT Latch0.0    ; Mouse L Sel
MMSel   BIT Latch0.1    ; Mouse M Sel
MRSel   BIT Latch0.2    ; Mouse R Sel
MOff    BIT Latch0.3    ; Mouse Off
MC      BIT Latch0.4    ; Mouse Click
MDC     BIT Latch0.5    ; Mouse D-Click
MS      BIT Latch0.6    ; Mouse Switch
Next    BIT Latch0.7    ; Next

Latch1: DS 1            ; Ausgabelatch 1
A1Sel   BIT Latch1.0    ; A1 Sel
A2Sel   BIT Latch1.1    ; A2 Sel
A3Sel   BIT Latch1.2    ; A3 Sel
MOn     BIT Latch1.3    ; Mouse On
MLT     BIT Latch1.4    ; Mouse L Taste
MMT     BIT Latch1.5    ; Mouse M Taste
MRT     BIT Latch1.6    ; Mouse R Taste
LBatt   BIT Latch1.7    ; low Batt

Port5:  DS 1            ; bitadressierbarer Speicher für P5
IR0     BIT Port5.0     ; IR D0
IR1     BIT Port5.1     ; IR D1
IR2     BIT Port5.2     ; IR D2
IR3     BIT Port5.3     ; IR D3
HM      BIT Port5.4     ; Hardware Mouse
HMin    BIT Port5.5     ; Hardware Mouse in

Error:  DS 1            ; Fehlerkennung
Ansch   BIT Error.0     ; Anschlußfehler
Bed1    BIT Error.1     ; Bedienungsfehler1 - Mod 3 E2 länger on als acctime
```

```
Bed2    BIT Error.2     ; Bedienungsfehler2 - Mod 3 E2 beim zweiten mal länger on als acctime
Fat1    BIT Error.3     ; Fatal1 - switchout nicht 1...4 (unmögliches Gerät)
Fat2    BIT Error.4     ; Fatal2 - mousesel nicht 1...4 (unmögliche Bedienungsart)
Fat3    BIT Error.5     ; Fatal3 - Mod nicht 1...5 (unmöglicher Modus, Mod6...8 nur im Mousemod)
Fat4    BIT Error.6     ; Fatal4 - attention nicht 0...7 (unbekannte Message)
Fat5    BIT Error.7     ; Fatal5 - mouseout nicht 1...4 (unmöglicher Ausgang)
;Fat6   Fat1+Fat2       ; Fatal6 - unmögliche Stufe in seriellem Interrupt
;Fat7   Fat1+Fat3       ; Fatal7 - unmögliche Kennung in seriellem Interrupt
;Fat8   Fat1+Fat4       ; Fatal8 - unmöglicher Befehl in seriellem Interrupt

lError: DS 1            ; letzer Fehler vor Reset
LAnsch  BIT Error.0     ; Anschlußfehler
LBed1   BIT Error.1     ; Bedienungsfehler1 - Mod 3 E2 länger on als acctime
LBed2   BIT Error.2     ; Bedienungsfehler2 - Mod 3 E2 beim zweiten mal länger on als acctime
LFat1   BIT Error.3     ; Fatal1 - switchout nicht 1...4 (unmögliches Gerät)
LFat2   BIT Error.4     ; Fatal2 - mousesel nicht 1...4 (unmögliche Bedienungsart)
LFat3   BIT Error.5     ; Fatal3 - Mod nicht 1...5 (unmöglicher Modus, Mod6...7 nur im Mousemod)
LFat4   BIT Error.6     ; Fatal4 - attention nicht 0...5 (unbekannte Message)
LFat5   BIT Error.7     ; Fatal5 - mouseout nicht 1...4 (unmöglicher Ausgang)

Misc:   DS 1            ; verschiedene Bits zusammengefaßt
MMon    BIT Misc.0      ; ist 1 wenn Mousemode aktiv
PrOn    BIT Misc.1      ; ist 1 wenn im Programmode
tcar    BIT Misc.2      ; ist 1 wenn ticks übergelaufen ist (ticks carry)
TelOn   BIT Misc.3      ; ist 1 wenn im Mod5 im Telefonmodus
oHM     BIT Misc.4      ; enthält "alten" HM
EAnsch  BIT Misc.5      ; Ansch Fehler schon bekannt
EBed1   BIT Misc.6      ; Bed1 Fehler schon bekannt
EBed2   BIT Misc.7      ; Bed2 Fehler schon bekannt

Misc2:  DS 1            ; nochmals verschiedene Bits
abusy   BIT Misc2.0     ; ist 1 wenn Attentiontest busy
wready  BIT Misc2.1     ; ist 1 wenn Timer abgelaufen
oA1     BIT Misc2.2     ; Speicher für A1 im Progmode
oScOn   BIT Misc2.3     ; Speicher für "alten" Scanmode

SerB:   DS 1            ; verschiedene Bits rund um die serielle Schnittstelle
STel    BIT SerB.0      ; ist 1 wenn Telefonschnittstelle angemeldet
SPC     BIT SerB.1      ; ist 1 wenn PC angemeldet
STFree  BIT SerB.2      ; ist 0 wenn Serielle auf Bestätigung des Notrufes wartet
SAl     BIT SerB.3      ; ist 1 wenn Serielle im Notrufmodus
S1      BIT SerB.4      ; ist 1 wenn auf 2. Byte gewartet wird
SAlarm  BIT SerB.5      ; ist 1 wenn Alarmeinrichtung angemeldet
SInit   BIT SerB.6      ; ist 1 wenn selbstausgelöste Initialisierung läuft
```

Hier werden alle Adressen definiert, wo der Zugriff auch auf einzelne Bits nötig ist. Einzelne Bits habe ich nicht über die entsprechende Direktive definiert, sondern in Bytes zusammengefaßt, da dadurch eine Abfrage über die serielle Schnittstelle leichter möglich ist.

9.7.1.5 Byteadressen ohne Bitadressen

```
 ; Byte Direktadressen (ohne Bitadressen)
 ; ======================================

        DSEG AT 48

 ; Konstante

Prell:  DS 1            ; Entprellticks
ctime:  DS 1            ; Dauer eines Clicks
dtime:  DS 1            ; Dauer zwischen den beiden Clicks eines Doppelclicks

 ; Variable

ticks:  DS 1            ; Zeitimpulse (Zeitmessung)
ticks2: DS 1            ; Zeitimpulse (relative Zeitmessung)
wait:   DS 1            ; Warteticks
otime:  DS 1            ; Länge des einfachen Ausgangsimpulses
atime:  DS 1            ; Länge der Referenzzeit
ktime:  DS 1            ; Länge Karrenzzeit
stime:  DS 1            ; Zeit zwischen dem automatischen Weiterschalten
stimem: DS 1            ; Speicher zur Zeitmessung für das Scannen
```

```
ftime:  DS 1            ; Speicher zur Zeitmessung für's Blinken
btime:  DS 1            ; Speicher zur Zeitmessung für's Batterietesten
Stufe:  DS 1            ; aktuelle Stufe der Protokollverhandlungen
Kenn:   DS 1            ; Kennung des Partnergerätes
Befehl: DS 1            ; aktueller Befehl
Speich: DS 1            ; angesprochene Speicherstelle
ALO:    DS 1            ; Speicher der LEDs für Fehlermeldungen
TimI:   DS 1            ; "alte" Tasteneinstellung
ABatt:  DS 1            ; aktueller Batteriewert
RAMOK:  DS 1            ; enthält einen OK-Wert

 ; Buffer

BLO:    DS 1            ; für Latch 0
BL1:    DS 1            ; für Latch 1
LLO:    DS 1            ; für Latch 0
LL1:    DS 1            ; für Latch 1
SerBuf: DS 1            ; aktueller Empfangswert
ODIP:   DS 1            ; alte DIP-Einstellungen
```

Auch Konstante habe ich als Variable definiert, damit diese — wenn es unbedingt nötig ist — über die serielle Schnittstelle verändert werden können.

9.7.1.6 externe Adressen

```
 ; externe Byteadressen (für Latches)
 ; ==================================

        XSEG AT 0

XL0     XDATA NOT 1000000000000000b     ; Latch 0 (A15)
XL1     XDATA NOT 0100000000000000b     ; Latch 1 (A14)
```

Die Adressen wurden mit `NOT` definiert, da sich so eine einfachere Hardware zur Selektion der Latches[6] ergibt.

9.7.1.7 Bitaliases

```
 ; Bitaliases (keine Definitionen)
 ; ===============================

        BSEG

DIP     EQU P1          ; DIP = P1
Karr0   BIT P1.0        ; Dip 0 - Karr0 (Karrenzzeit Bit 0)
Karr1   BIT P1.1        ; Dip 1 - Karr1 (Karrenzzeit Bit 1)
out3On  BIT P1.2        ; Dip 2 - out 3 on/off
out2On  BIT P1.3        ; Dip 3 - out 2 on/off
MoOn    BIT P1.4        ; Dip 4 - mouse on/off (generell)
ScOn    BIT P1.5        ; Dip 5 - scan on/off
MoMOn   BIT P1.6        ; Dip 6 - Mouse Mitte on/off
PrgOn   BIT P1.7        ; Dip 7 - progable on/off

Al      BIT P3.4        ; Alarm
A3      BIT P3.5        ; Ausgang 3

E1      BIT P4.0        ; Eingang 1
E1in    BIT P4.1        ; Eingang 1 in
E2      BIT P4.2        ; Eingang 2
E2in    BIT P4.3        ; Eingang 2 in
E3      BIT P4.4        ; Eingang 3
E3in    BIT P4.5        ; Eingang 3 in
A1      BIT P4.6        ; Ausgang 1
A2      BIT P4.7        ; Ausgang 2
```

[6] für eine genauere Erläuterung siehe Kapitel 11.6

Ich habe auch die Portbits mit eigenen Namen versehen. Dies hat zwei Gründe: Erstens kann man sich die Namen beim Programmieren leichter merken (auch das Lesen des Programmes wird so einfacher) und zweites kann man auf diese Weise zukünftige Hardwareänderungen leichter im Programm berücksichtigen.

9.7.1.8 Kennwerte

```
 ; Kennwerte
 ; =========

        DSEG            ; nur zur Sicherheit

; Attention-Werte

A_OK    EQU 0           ; alles OK
A_Mod   EQU 1           ; Moduswechsel (Mod1 ... Mod6)
A_Prog  EQU 2           ; Programmieraufforderung
A_HW    EQU 3           ; Hardware-Wechsel Mousemode/Switchmode
A_SWSM  EQU 4           ; Software-Wechsel von Switchmode in Mousemode
A_SWMS  EQU 5           ; Software-Wechsel von Mousemode in Switchmode
A_Cold  EQU 6           ; ColdReset
A_Warm  EQU 7           ; WarmReset

; Modis

M_NEW   EQU 0           ; Modus neu feststellen (z.B. E1 wurde abgesteckt)
M_1     EQU 1           ; Single (nur E1)
M_2     EQU 2           ; Single + Select (E1 + E3)
M_3     EQU 3           ; Double (nur E2)
M_4     EQU 4           ; Double + Select (E2 + E3)
M_5     EQU 5           ; IR
M_6     EQU 6           ; Scanmode Single (nur im Mousemode erlaubt)
M_7     EQU 7           ; Scanmode Double (nur im Mousemode erlaubt)
M_8     EQU 8           ; Scanmode IR (nur im Mousemode erlaubt)

; mout-Werte (mouse out)

links   EQU 1
mitte   EQU 2
rechts  EQU 3
M_Off   EQU 4

; msel-Werte (mouse sel)

click   EQU 1
dclick  EQU 2
switch  EQU 3
nextm   EQU 4

; sout-Werte (switch out)

G1      EQU 1
G2      EQU 2
G3      EQU 3
M_On    EQU 4

; Serielle Codes

SRead   EQU 11000000b   ; PC Read
SWrite  EQU 00011000b   ; PC Write
SReset  EQU 00000011b   ; PC Reset
SWarm   EQU 0           ; Warmstart
SCold   EQU 0FFh        ; ColdStart
SEsc    EQU 0FFh        ; Esc
SOK     EQU 0           ; OK
SNo     EQU 1           ; Fehler
SNot    EQU 10101010b   ; Alarm-Code
Stufe0  EQU 0           ; Stufe 0 im Protokoll
Stufe1  EQU 1           ; Stufe 1 im Protokoll
Stufe2  EQU 2           ; Stufe 2 im Protokoll
SBad    EQU 3           ; Unkompatibilität im Protokoll
MaxCode EQU 22          ; Maximale Codenummer
C_P1    EQU 1           ; Code für P1
C_P3    EQU 2           ; Code für P3
C_P4    EQU 3           ; Code für P4
```

```
C_P5    EQU 4           ; Code für P5
C_ABatt EQU 22          ; Code für ABatt

; Geräte

SG_MS   EQU 0           ; Multiswitch selbst
SG_TS   EQU 1           ; Telefonschnitstelle
SG_PC   EQU 2           ; PC-Gerätenummer

; RAM Test

ROK     EQU 10101010b   ; Testwert zur Erkennung von RAM-Ausfällen

; Sonder-Zeiten

flash   EQU 42          ; Blinkzeit (ca. 0.5s)
IniWait EQU 42          ; Wartezeit auf serielle Antwort (ca. 0.5s)
Battest EQU 85          ; Batterietestzeit (ca. 1s)

; diverse

KBit    EQU 00000011b           ; Karr-Bits
MBit    EQU 00001111b           ; Mouse-LED-Bits (im Latch 0)

; Basis

Bit0    EQU 00000001b
Bit1    EQU 00000010b
Bit2    EQU 00000100b
Bit3    EQU 00001000b
Bit4    EQU 00010000b
Bit5    EQU 00100000b
Bit6    EQU 01000000b
Bit7    EQU 10000000b

nBit0   EQU NOT Bit0
nBit1   EQU NOT Bit1
nBit2   EQU NOT Bit2
nBit3   EQU NOT Bit3
nBit4   EQU NOT Bit4
nBit5   EQU NOT Bit5
nBit6   EQU NOT Bit6
nBit7   EQU NOT Bit7
```

Nahezu alle Kennwerte wurden über `EQU` definiert. Damit sind erstens Änderungen und Erweiterungen leichter und zweites ist das Programm leichter lesbar.

9.7.1.9 Konstante

```
 ; Konstante
 ; =========

; ca. 85 ticks = 1 s

KPrell  EQU 10          ; Entprellticks
Kctime  EQU 10          ; Dauer eines Clicks
Kdtime  EQU 10          ; Dauer zwischen den beiden Clicks eines Doppelclicks
Kotime  EQU 30          ; Dauer des einfachen Ausgangsimpulses
Katime  EQU 80          ; Länge der Referenzzeit (1s)
Kktime  EQU 40          ; Länge der Karrenzzeit (0.5s)
Kstime  EQU 70          ; Zeit zwischen dem automatischen Weiterschalten

 ; Analogwerte

BSchw   EQU 170         ; Schwelle für Batteriealarm

 ; IR-Belegung

I_Pl    EQU 1           ; nächste Taste
I_Go    EQU 4           ; Taste auslösen

I_Tel   EQU 15          ; Schalter zwischen Telefon und MultiSwitch
```

Hier wurden alle Konstante, die für Entscheidungen und die Grundeinstellung benötigt

werden definiert. Es ist gerade in diesem Zusammenhang sehr wahrscheinlich, daß sich diese Werte nach eingehenderen Tests verändern.

9.7.1.10 Interruptstartadressen

```
 ; Interrupt Startadressen
 ; =======================

        CSEG

Reset   CODE 0000H      ; Reset
X0      CODE 0003H      ; External interrupt 0
T00     CODE 000BH      ; Timer 0 overflow
X1      CODE 0013H      ; External interrupt 1
T01     CODE 001BH      ; Timer 1 overflow
SIO0    CODE 0023H      ; SIO0 (UART)
SIO1    CODE 002BH      ; SIO1 (I2C)
CT0     CODE 0033H      ; T2 capture 0
CT1     CODE 003BH      ; T2 capture 1
CT2     CODE 0043H      ; T2 capture 2
CT3     CODE 004BH      ; T2 capture 3
ADC     CODE 0053H      ; ADC completion
CM0     CODE 005BH      ; T2 compare 0
CM1     CODE 0063H      ; T2 compare 1
CM2     CODE 006BH      ; T2 compare 2
T02     CODE 0073H      ; T2 overflow
```

Aus unerklärlichen Gründen sind die Interruptstartadressen nicht im Assembler vordefiniert, sodaß ich das hier nachholen mußte.

9.7.2 die Unterroutinen

Das Programm habe ich nach einem 4–Schicht–System aufgebaut (siehe Abbildung 9.5).

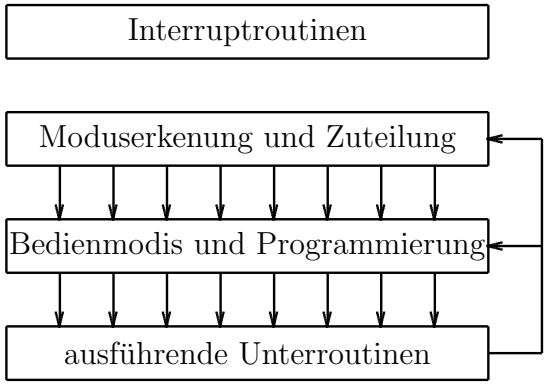

Abbildung 9.5: Darstellung des schichtweisen Aufbaus des Programmes TheProg

Die oberste Schichte stellt die Routine `att_tst` (Attention–Test) dar. In dieser werden die von den Interruptroutinen oder Modis gesetzten Messages ausgewertet; neue Modis werden hier erkannt und die entsprechenden Module angesprungen.

In der nächsten Schichte befinden sich alle ausführenden Module. Für jeden Bedienmodus (Mod1 bis Mod8) ist ein eigenes Modul vorhanden, zusätzlich ist der Programmiermodus ebenfalls ein eigenes Modul. Diese Unterprogramme sind die eigentlichen Ausführungsorgane des Programmes, die, spezifisch für den Bedienungsmodus, Aktionen durchführen. Das Programmiermodul hat eine Sonderstellung, da nahezu alle Interrupts abgeschaltet werden, um die Programmierung nicht zu stören und die Kontrolle über die Ausgabe–LEDs zu erhalten. Bei jedem Durchlauf eines Moduls wird geprüft, ob sich im Laufe der Modulausführung oder durch eine Interruptroutine eine Message ergeben hat; ist dies der Fall wird die Programmkontrolle an die oberste Schichte (`att_tst`) zurückgegeben.

Die dritte Stufe besteht aus den vielen Unterroutinen, die die „eigentliche“ Arbeit verrichten. Die Module rufen diese auf, um einzelne Teilarbeiten (z.B. nächster Ausgang, Impuls ausgeben, Zeit abwarten, ...) zu verrichten. Alle Unterroutinen werden über den `call`–Befehl aufgerufen und können daher mit `ret` verlassen werden. Manche Unterroutinen holen sich aus „globalen“ Variablen Informationen über den aktuellen Zustand der Hardware bzw. des Programmiermoduses, um z.B. den nächsten erlaubten Ausgang finden zu können.

Die vierte Schicht des Programmes besteht aus den Interruptroutinen. Diese sind völlig (bis auf wenige Ausnahmen in den Scanmodis) von den restlichen Routinen getrennt und kommunizieren nur über wenige Informationsbits (z.B. `abusy, tcar, ...`) und Messages mit dem Rest des Programmes.

9.7.2.1 Tabelle der Interruptvektoren

```
; Reset + Interrupts
; ==================

       CSEG AT Reset   ; Reset

       ajmp Init       ; zum Initialisierungscode

; ---------------------

       CSEG AT X0      ; IR-Ready

       ajmp Int00      ; zur Serviceroutine

; ---------------------

       CSEG AT T00     ; gesamte Zeitsteuerung

       ajmp Timer0     ; zur Serviceroutine

; ---------------------

       CSEG AT X1      ; PowerOff

       ajmp Int01      ; zur Serviceroutine

; ---------------------

       CSEG AT T01     ; Timer 1 overflow

       reti            ; unused

; ---------------------
```

```
       CSEG AT SIO0    ; serielle Schnittstelle
       ajmp Ser        ; zur Serviceroutine
; ---------------------
       CSEG AT SIO1    ; I2C
       reti            ; unused
; ---------------------
       CSEG AT CT0     ; T2 capture 0
       reti            ; unused
; ---------------------
       CSEG AT CT1     ; T2 capture 1
       reti            ; unused
; ---------------------
       CSEG AT CT2     ; T2 capture 2
       reti            ; unused
; ---------------------
       CSEG AT CT3     ; T2 capture 3
       reti            ; unused
; ---------------------
       CSEG AT ADC     ; ADC fertig
       reti            ; unused - ADC wird ohne Interrupt verarbeitet
; ---------------------
       CSEG AT CM0     ; T2 compare 0
       reti            ; unused
; ---------------------
       CSEG AT CM1     ; T2 compare 1
       reti            ; unused
; ---------------------
       CSEG AT CM2     ; T2 compare 2
       reti            ; unused
; ---------------------
       CSEG AT TO2     ; T2 overflow
       reti            ; unused
```

In diesem Teil werden alle Interruptvektoren initialisiert.

9.7.2.2 Interruptserviceroutinen

9.7.2.2.1 externer Interrupt 0

```
; Interruptserviceroutinen
; ========================

Int00:  push PSW               ; PSW sichern
```

```
        push Acc                ; A sichern
        clr RS1                 ; auf Bank 1 schalten
        setb RS0
        push R0_0               ; Register 0 in Bank 0 sichern
        push R1_0               ; Register 1 in Bank 0 sichern
        mov R0,P5               ; Port 5 ins Register 0 zur Weiterbearbeitung
        anl R0_1,#00001111b     ; IR-Bits herausschneiden
        mov A,Modus             ; kein Zugang über Register!
        cjne A,#M_5,Int0_0A     ; springe wenn nicht im IR-Modus
Int0_0C:jb TelOn,Int0_A         ; springe wenn im Telefonmodus
        cjne R0,#I_Tel,Int0_B   ; springe wenn Code != I_Tel
        jnb STel,Int_End        ; springe wenn kein Telefon da
        setb TelOn              ; Telefonmodus ein
        ajmp Int_End            ; das war's

Int0_0A:cjne A,#M_8,Int0_A      ; springe wenn in keinem IR-Modus
        sjmp Int0_0C            ; geht doch klar

Int0_B: cjne R0,#I_Go,Int0_E    ; springe wenn Code != I_Go
        clr RS0                 ; auf Bank 0 wechseln
        acall Puls              ; Ausgang aktivieren
        sjmp Int_End            ; das war's

Int0_E: cjne R0,#I_Pl,Int_End   ; springe wenn Code != I_Pl
        clr RS0                 ; auf Bank 0 wechseln
        acall Nextout           ; nächster Ausgang
        sjmp Int_End            ; das war's

Int0_A: cjne R0,#I_Tel,Int0_C   ; springe wenn Code != I_Tel
        clr TelOn               ; Telefonmodus aus
        ajmp Int_End            ; das war's

Int0_C: jnb STel,Int_End        ; springe wenn keine TelSchnittstelle da ist
        jnb STFree,Int_End      ; springe wenn SerTransmit nicht frei
        mov S0BUF,R0            ; Code an Serielle weiterleiten
        sjmp Int_End            ; das war's

Int_End:pop R1_0                ; Register 1 in Bank 0 restaurieren
        pop R0_0                ; Register 0 in Bank 0 restaurieren
        pop Acc                 ; A restoren
        pop PSW                 ; PSW restoren
        clr IE0                 ; alle Codes während des Interrupts ignorieren
        reti
```

Zunächst wird auf eine andere Registerbank, wie bei allen Interruptroutinen, geschaltet und wichtige Register werden am Stack gesichert. Aus diesem Grund wurde von mir eine Ausstiegsroutine am Ende geschrieben um immer diese Sicherungsvorgänge umzukehren.

Die Routine unterscheidet die Fälle des normalen Infrarotmoduses, des Telefonmoduses und des Moduses ohne Beteiligung der IR–Schnittstelle. Im normalen Infrarotmodus werden die zwei Eingangssensoren simuliert, im Telefonmodus werden die Codes zur seriellen Schnittstelle weitergeleitet (außer der Code bedeutet die Umschaltung auf den normalen Infrarotmodus); sonst wird der Code ignoriert.

9.7.2.2.2 externer Interrupt 1

```
Int01:  clr EA                  ; keine Interrupts mehr
        mov P0,#00h             ; alle Ports löschen
        mov P1,#00h
        mov P2,#00h
        mov P3,#00h
        mov P4,#00h
Int1_A: mov PCON,#00000010b     ; in den PowerDownModus
        sjmp Int1_A             ; nur zur Sicherheit
```

Hier werden alle Interrupts gestoppt, alle Ports auf 0 gesetzt und der Power–Down–Modus gestartet.

9.7.2.2.3 Timer 0

```
Timer0: push PSW                ; PSW sichern
        push Acc                ; A sichern
        setb RS1                ; Registerbank 2 wählen
        clr RS0
        mov TL0,#207            ; Lowbyte reloaden
        mov TH0,#213            ; Highbyte reloaden

                                ; <--- alle 0.0117s = 1 tick

        inc ticks               ; Zeitzählung um 1 erhöhen
        mov A,ticks             ; ticks in A zur overflow-Kontrolle
        jnz Tim_A               ; springe wenn kein Überlauf stattgefunden hat
        setb tcar               ; Überlaufbit setzen
Tim_A:  inc ticks2              ; Zeitzählung 2 um 1 erhöhen
        djnz wait,Tim_B         ; Wartezähler um 1 erniedrigen und springen wenn != 0
        setb wready             ; Fertigbit setzen
Tim_B:  jb abusy,Tim_E          ; keine Messages wenn der Modus gerade bestimmt wird
        mov A,P4                ; Port 4 in A zum Vergleich
        anl A,#00101010b        ; entscheidende Bits herausschneiden
        mov Port5,P5            ; Port 5 aktualisieren
        mov C,HMin              ; HMin auch ins A
        mov Acc.7,C
        cjne A,TimI,Tim_C       ; springe wenn "alte" Stellung != "neue"
        sjmp Tim_D              ; war nichts zu tun

Tim_C:  mov TimI,A              ; neuen Stand sichern
        mov atten,#A_Mod        ; Message senden
        sjmp Tim_E              ; zum nächsten Sammelpunkt (immer nur 1 Message!)

Tim_D:  jb HMin,Tim_H           ; springe wenn HW-switch nicht in
        jb HM,Tim_F             ; springe wenn HM == 1
        jb oHM,Tim_G            ; HM == 0 | springe wenn oHM == 1
        sjmp Tim_H              ; HM == oHM == 0

Tim_F:  jb oHM,Tim_H            ; HM == 1 | springe wenn oHM == 1
Tim_G:  mov atten,#A_HW         ; HM != oHM | Message senden
        mov C,HM                ; HM in oHM zum nächsten Vergleich
        mov oHM,C
        sjmp Tim_E              ; zum nächsten Sammelpunkt (immer nur 1 Message!)

Tim_H:  jb ScOn,Tim_H1          ; springe wenn ScOn == 1
        jb oScOn,Tim_H2         ; ScOn == 0 | springe wenn ScOn == 1
        sjmp Tim_H3             ; ScOn == 0 && oScOn == 0 | war nichts zu tun

Tim_H1: jb oScOn,Tim_H3         ; ScOn == 1 | springe wenn oScOn == 1 -> ScOn == oScOn
Tim_H2: cpl oScOn               ; ScOn != oScOn | oScOn verändern -> oScOn == ScOn
        mov atten,#A_Mod        ; Modus neu bestimmen
        sjmp Tim_E              ; zum nächsten Sammelpunkt (immer nur 1 Message!)

Tim_H3: jb PrgOn,Tim_E          ; war nichts zu tun | springe wenn Prog==off
        mov atten,#A_Prog       ; Message senden
Tim_E:  jnb Al,Tim_I            ; springe wenn kein Alarm
        jnb SAlarm,Tim_I        ; springe wenn keine Alarmeinrichtung da
        jnb SAl,Tim_J           ; springe wenn Alarm noch nicht bekannt
        jb STFree,Tim_K         ; springe wenn Antwort schon eingetroffen
        mov S0Buf,#SNot         ; Alarm-Code senden
        sjmp Tim_K              ; auf Antwort warten

Tim_J:  clr STFree              ; Serielle auf Biegen und Brechen belegen
        mov Befehl,#0           ; bisherigen Befehl löschen
        mov S0Buf,#SNot         ; Alarm-Code senden (Zustand der Seriellen mißachtend!)
        setb SAl                ; Alarm bekannt
        sjmp Tim_K              ; auf Antwort warten

Tim_I:  clr SAl                 ; kein Alarm
        setb STFree             ; ohne Alarm ist die Serielle auch wieder frei!
Tim_K:  djnz stimem,Tim_L       ; springe wenn nicht Scannen

                                ; <--- alle stime

        mov stimem,stime        ; Zähler neu laden
        jnb PrOn,Tim_M          ; springe wenn nicht im ProgMode
        jb Ansch,Tim_L          ; springe wenn Fehler angezeigt wird
        mov A,Latch0            ; Latch0 in A für Veränderung
        rl A                    ; LED "rotieren" lassen
        mov Latch0,A            ; Latch0 zurückschreiben
        acall mL0               ; Latch 0 aktualisieren
        sjmp Tim_L              ; Scannen erledigt
```

```
Tim_M:  jnb MMon,Tim_L          ; springe wenn nicht im Mousemode
        jb ScOn,Tim_L           ; springe wenn nicht im Scanmode
        clr RS1                 ; Registerbank 0 wählen
        acall Nextmouse         ; nächste Aktivierungsart wählen
        setb RS1                ; Registerbank 2 wählen
Tim_L:  djnz ftime,Tim_N        ; springe wenn nicht Blinken

                                ; <--- alle flash ticks

        mov ftime,#flash        ; ftime reloaden
        mov A,Error             ; Error für Kontrolle in A
        jz Tim_R                ; springe wenn kein Fehler
        jnb Ansch,Tim_O         ; springe wenn Fehler != Ansch
        jb EAnsch,Tim_P         ; springe wenn Fehler schon bekannt
        mov AL0,#11110000b      ; setzen eines LED-Musters
        setb EAnsch             ; Fehler ab jetzt bekannt
        sjmp Tim_Q              ; Anzeigen!

Tim_P:  mov A,AL0               ; AL0 in A für Komplement
        cpl A                   ; Komplement
        mov AL0,A               ; wieder zurückspeichern
        sjmp Tim_Q              ; Anzeigen

Tim_O:  jnb Bed1,Tim_S          ; springe wenn Fehler != Bed1
        jb EBed1,Tim_P          ; springe wenn Fehler schon bekannt
        setb EBed1              ; Fehler ab jetzt bekannt
        mov AL0,#11001100b      ; setzen eines LED-Musters
        sjmp Tim_Q              ; Anzeigen

Tim_S:  jnb Bed2,Tim_T          ; springe wenn Fehler != Bed2
        jb EBed2,Tim_P          ; springe wenn Fehler schon bekannt
        setb EBed2              ; Fehler ab jetzt bekannt
        mov AL0,#10101010b      ; setzen eines LED-Musters
        sjmp Tim_Q              ; Anzeigen

Tim_T:  ljmp FatalError         ; dann kann's nur ein fataler sein

Tim_Q:  mov DPTR,#XL0           ; Adresse des Latches 0 laden
        mov A,AL0               ; Wert des Latches 0 laden
        movx @DPTR,A            ; ins Latch 0 speichern
        sjmp Tim_N              ; das war's

Tim_R:  acall mL0               ; Latch 0 aktualisieren
        clr EAnsch              ; alle Fehler unbekannt
        clr EBed1               ;
        clr EBed2               ;
Tim_N:  djnz btime,Tim_X        ; springe wenn nicht Zeit zum Batterietest

                                ; <--- alle Battest ticks

        mov btime,#Battest      ; Zeit reloaden
        jb PrOn,Tim_X           ; kein Test während des Programmierens
        anl ADCON,#nBit0        ; ADC-Eingang ADC6 wählen
        orl ADCON,#Bit3         ; ADC starten
Tim_U:  mov A,ADCON             ; ADCON ins A zur Bitabfrage
        jnb Acc.4,Tim_U         ; Warten auf ADC-Wandlung
        anl ADCON,#nBit4        ; Interruptflag löschen
        mov A,ADCH              ; Ergebnis in A zum Vergleich
        mov ABatt,A             ; Ergebnis sichern
        cjne A,#BSchw,Tim_V     ; springen wenn R0 BSchw verschieden
        sjmp Tim_W              ; == Schwelle

Tim_V:  jnc Tim_X               ; springen wenn Batteriewert > Schwelle (OK!)
Tim_W:  clr LBatt               ; LowBatt-LED on
        acall mL1               ; Latch 1 aktualisieren
Tim_X:  pop Acc                 ; A restoren
        pop PSW                 ; PSW restoren
        reti                    ; auf ein Neues
```

Nach dem Sichern wichtiger Register wird der Timer reloaded, um eine Intervallzeit von 0.0117s zu erreichen; danach werden die Zeitvariablen für die Zeitmessung und die Warteroutine erhöht. Darauf folgt die Überprüfung der Eingänge. Sollte sich eine Veränderung ergeben haben, so wird eine entsprechende Message gesendet. Beim Auftreten eines Alarmrufes wird die serielle Schnittstelle belegt und der Notruf weitergeleitet.

Alle `stime` wird im Programmiermodus die einzige LED rotiert, in einem Scanmodus der nächste Ausgang angewählt, sonst wird nichts verändert, alle `ftime` werden die Fehlermeldungen abgefragt und wenn nötig angezeigt, alle `btime` wird ein Batterietest durchgeführt und wenn nötig die Batteriealarmanzeige aktiviert.

9.7.2.2.4 serielle Schnittstelle

```
Ser:    push PSW                ; PSW sichern
        push Acc                ; A sichern
        setb RS0                ; Registerbank 1 wählen
        clr RS1
        jbc TI,Ser_A            ; Transmit / springe und lösche
        jbc RI,Ser_B            ; Receive / springe und lösche
        ajmp Ser_End            ; keine Ahnung was es war, aber zum Ende

Ser_A:  ajmp Ser_End            ; zum Ende

Ser_0C: ajmp Ser_C              ; Verlängerung des relativen Sprunges

Ser_B:  mov SerBuf,S0Buf        ; Empfangswert sichern
        jb SInit,Ser_BA         ; springe wenn bei Initialisierung
        jb STel,Ser_0C          ; springe wenn Telefonschnittstelle angemeldet
        jnb SPC,Ser_D           ; springe wenn PC nicht angemeldet
        ajmp Ser_DD
Ser_D:  mov R0,Stufe            ; Stufe in R0 für folgenden Vergleich
        cjne R0,#Stufe0,Ser_F   ; springe wenn nicht in Stufe0 (Grundstufe)
        mov A,SerBuf            ; aktuellen Code in A für Vergleich
        cjne A,#SEsc,Ser_En0    ; springen wenn nicht Esc (ignoriere alles andere)
        mov Stufe,#Stufe1       ; nächste Stufe erreicht
Ser_En0:ajmp Ser_End            ; das wars für's erste

Ser_BA: mov R0,Stufe            ; Initialisierung | Stufe in R0 zum Vergleich
        cjne R0,#Stufe0,Ser_BB  ; springe wenn Stufe != Stufe0
        mov A,SerBuf            ; aktuellen Code in A zum Vergleich
        jnz Ser_BC              ; springe wenn Antwort != OK
        mov Stufe,#Stufe1       ; Stufe1 erreicht
        ajmp Ser_End            ; warten auf Kennung

Ser_BC: mov Stufe,#SBad         ; untauglich
        ajmp Ser_End            ; warten auf Kennung

Ser_BB: cjne R0,#Stufe1,Ser_BD  ; springe wenn Stufe != Stufe1
        mov Kenn,SerBuf         ; Kennung speichern
        mov A,SerBuf            ; Kennung in A zum Vergleich
        cjne A,#SG_TS,Ser_BE    ; springe wenn Kennung != SG_TS
        setb STel               ; Telefonsubgerät anmelden
        sjmp Ser_BF             ; Telefonschnittstelle ist OK!

Ser_BE: cjne A,#SG_PC,Ser_BG    ; springe wenn Kennung != SG_PC
        setb SPC                ; PC anmelden
Ser_BF: mov S0Buf,#SOk          ; OK senden
        sjmp Ser_BD             ; Ende des Init-Modes

Ser_BG: mov S0Buf,#SNo          ; unbekanntes Gerät
Ser_BD: mov Stufe,#Stufe0       ; Stufe zurücksetzen
        setb wready             ; Warten aus
        ajmp Ser_End

Ser_F:  cjne R0,#Stufe1,Ser_L   ; != Stufe0 | springe wenn nicht in Stufe1
        mov Kenn,SerBuf         ; Kennung speichern
        mov A,SerBuf            ; Kennung in A zum Vergleich
        cjne A,#SG_TS,Ser_G     ; springe wenn nicht Telefonschnittstelle
        sjmp Ser_I              ; Telefonschnittstelle ist ok!

Ser_G:  cjne A,#SG_PC,Ser_H     ; springe wenn nicht PC
Ser_I:  mov S0Buf,#SOk          ; Gerät ok! | positive Antwort senden
Ser_J:  jnb TI,Ser_J            ; Warten auf Übertragung
        clr TI                  ; Fertig-Bit löschen
        mov S0Buf,#SG_MS        ; eigene Kennung senden
        mov Stufe,#Stufe2       ; nächste Stufe erreicht
        ajmp Ser_End            ; das war's in der Stufe

Ser_H:  mov S0Buf,#SNo          ; falsches Gerät! | negative Antwort senden
Ser_K:  jnb TI,Ser_K            ; Warten auf Übertragung
        clr TI                  ; Fertig-Bit löschen
```

```
        mov S0Buf,#SG_MS        ; eigene Kennung senden
        mov Stufe,#Stufe0       ; Stufe auf Anfang zurücksetzen
        ajmp Ser_End            ; das war's für diesen Versuch

Ser_L:  cjne R0,#Stufe2,Ser_M   ; != (Stufe0 || Stufe1) | springe wenn nicht Stufe 2
        mov A,SerBuf            ; SerBuf ins A für folgende Entscheidung
        jz Ser_N                ; springe wenn Antwort=Ok
        mov Stufe,#Stufe0       ; Stufe zurücksetzen
        ajmp Ser_End            ; alles vergessen

Ser_N:  mov A,Kenn              ; Kennung in A für folgende Entscheidung
        cjne A,#SG_TS,Ser_O     ; springe wenn Gerät != Telefonschnittstelle
        setb STel               ; Kennung für Telefonschnittstelle setzen
        setb SAlarm             ; Kennung für Alarmgerät setzen
        mov Stufe,#Stufe0       ; Stufe zurücksetzen
        ajmp Ser_End            ; Protokoll Ende

Ser_O:  cjne A,#SG_PC,Ser_P     ; springe wenn Gerät != PC
        setb SPC                ; Kennung für PC setzen
        mov Stufe,#Stufe0       ; Stufe zurücksetzen
        ajmp Ser_End            ; Protokoll Ende

Ser_P:  setb Fat1               ; Fatal-Error 7 setzen (ausnahmsweise 2 Bits!)
        setb Fat3
        ljmp FatalError         ; für immer verreist

Ser_M:  setb Fat1               ; Fatal-Error 6 setzen (außnahmesweise 2 Bits!)
        setb Fat2
        ljmp FatalError         ; keine Rückkehr!

Ser_C:  mov A,SerBuf            ; SerBuf in A für folgende Entscheidung
        jb STFree,Ser_Q         ; Telefonschnittstelle | springe wenn Serielle frei
        jnz Ser_CO              ; keine positive Antwort
        setb STFree             ; Serielle wieder frei
Ser_CO: ajmp Ser_End            ; das war's

Ser_Q:  cjne A,#SEsc,Ser_CO     ; springe wenn Code != SEsc
        clr STel                ; Telefonmodus beenden
        clr SAlarm              ; Alarmgerätkennung löschen
        mov Stufe,#Stufe1       ; auf Kennung warten
        ajmp Ser_End            ; das war's für's erste

Ser_DD: mov A,Befehl            ; Befehl in A zur folgenden Entscheidung
        jnz Ser_R               ; springe wenn nicht "kein Befehl"
        mov A,SerBuf            ; SerBuf in A für folgende Entscheidung
        cjne A,#SEsc,Ser_S      ; springe wenn Code != SEsc
        clr SPC                 ; PC-Modus beenden
        mov Stufe,#Stufe1       ; auf Kennung warten
        ajmp Ser_End            ; das war's für's erste

Ser_S:  cjne A,#SWrite,Ser_T    ; springe wenn Befehl != SWrite
        sjmp Ser_U              ; zum nächsten Sammelpunkt

Ser_T:  cjne A,#SRead,Ser_V     ; springe wenn Befehl != SRead
        sjmp Ser_U              ; zum nächsten Sammelpunkt

Ser_V:  cjne A,#SReset,Ser_W    ; springe wenn Befehl != SReset
Ser_U:  mov Befehl,A            ; Sammelpunkt | Befehl sichern
        setb S1                 ; auf 2. Byte warten
        mov SOBUF,#SOK          ; OK senden
        ajmp Ser_End            ; warten auf Argumente

Ser_W:  mov Befehl,#0           ; kein bekannter Befehl | Befehl nicht erkannt
        mov SOBUF,#SNo          ; Fehler senden
        ajmp Ser_End            ; auf neuen Befehl warten

Ser_R:  cjne A,#SWrite,Ser_X    ; irgendein Befehl | springe wenn Befehl != SWrite
        jnb S1,Ser_Y            ; springe wenn nicht 2. Byte eingetroffen
        clr S1                  ; 2. Byte empfangen
        mov A,SerBuf            ; SerBuf in A für folgenden Test
        acall Ser_Kon           ; Register kontrollieren
        jc Ser_Z                ; springe wenn Register <= MaxCode
Ser_AA: mov SOBUF,#SNo          ; Fehler senden
        mov Befehl,#0           ; kein Befehl
        ajmp Ser_End            ; auf neuen Befehl warten

Ser_Z:  cjne A,#C_P5,Ser_AB     ; springe wenn A != C_P5
        sjmp Ser_AA             ; P5 kann nicht beschrieben werden

Ser_AB: cjne A,#C_ABatt,Ser_ZA  ; springe wenn A != C_ABatt
        sjmp Ser_AA             ; ABatt kann nicht beschrieben werden
```

```
Ser_ZA: mov Speich,SerBuf       ; gültiges Register speichern
        mov S0BUF,#SOk          ; Ok senden
        ajmp Ser_End            ; warten auf zu speichernde Daten

Ser_Y:  mov A,Speich            ; Registernummer in A
        cjne A,#C_P1,Ser_KA     ; springen wenn Register != C_P1
        mov P1,SerBuf           ; Daten speichern
        sjmp Ser_KD             ; zum nächsten Sammelpunkt

Ser_KA: cjne A,#C_P3,Ser_KB     ; springen wenn Register != C_P3
        mov P3,SerBuf           ; Daten speichern
        sjmp Ser_KD             ; zum nächsten Sammelpunkt

Ser_KB: cjne A,#C_P4,Ser_KC     ; springen wenn Register != C_P4
        mov P4,SerBuf           ; Daten speichern
        sjmp Ser_KD             ; zum nächsten Sammelpunkt

Ser_KC: acall Ser_Reg           ; absolute Adresse aus Tabelle holen
        mov R0,A                ; Adresse in R0 für indirekte Adressierung
        mov @R0,SerBuf          ; Wert in angewähltes Register
Ser_KD: mov Befehl,#0           ; kein aktueller Befehl
        mov S0BUF,#SOk          ; Empfang bestätigen
        sjmp Ser_End

Ser_X:  cjne A,#SRead,Ser_AC    ; Befehl != SWrite | springe wenn Befehl != SRead
        mov A,SerBuf            ; Registernummer in A zur Kontrolle
        acall Ser_Kon           ; Nummer kontrollieren
        jc Ser_AD               ; springe wenn Register <= MaxCode
        mov S0BUF,#SNo          ; Fehler senden
        mov Befehl,#0           ; kein Befehl
        sjmp Ser_End            ; auf nächsten Befehl warten

Ser_AD: cjne A,#C_P1,Ser_IA     ; springe wenn Code != C_P1
        mov S0Buf,P1            ; Daten versenden
        sjmp Ser_ID             ; zum nächsten Sammelpunkt

Ser_IA: cjne A,#C_P3,Ser_IB     ; springe wenn Code != C_P3
        mov S0Buf,P3            ; Daten versenden
        sjmp Ser_ID             ; zum nächsten Sammelpunkt

Ser_IB: cjne A,#C_P4,Ser_IE     ; springe wenn Code != C_P4
        mov S0Buf,P4            ; Daten versenden
        sjmp Ser_ID             ; zum nächsten Sammelpunkt

Ser_IE: cjne A,#C_P5,Ser_IC     ; springe wenn Code != C_P5
        mov S0Buf,P5            ; Daten versenden
        sjmp Ser_ID             ; zum nächsten Sammelpunkt

Ser_IC: acall Ser_Reg           ; Registernummer in absolute Adresse wandeln
        mov R0,A                ; absolute Adresse in R0 für indirekte Adressierung
        mov S0BUF,@R0           ; Daten an Serielle
Ser_ID: mov Befehl,#0           ; kein Befehl
        sjmp Ser_End            ; auf nächsten Befehl warten

Ser_AC: cjne A,#SReset,Ser_AE   ; springe wenn Befehl != SReset
        mov A,SerBuf            ; Code in A für Vergleich
        mov Befehl,#0           ; kein Befehl
        cjne A,#SWarm,Ser_AF    ; springe wenn Code != SWarm
        mov S0BUF,#SOK          ; OK senden
Ser_WA: jnb TI,Ser_WA           ; Reset erst nach erfolgter OK-Meldung!
        clr TI                  ; Bit löschen
        mov atten,#A_Warm       ; Warm-Resetanforderung setzen
        sjmp Ser_End            ; das war's

Ser_AF: cjne A,#SCold,Ser_AG    ; springe wenn Code != SCold
        mov S0BUF,#SOK          ; OK senden
Ser_WB: jnb TI,Ser_WB           ; Reset erst nach erfolgter OK-Meldung!
        clr TI                  ; Bit löschen
        mov atten,#A_Cold       ; Cold-Resetanforderung setzen
        sjmp Ser_End            ; das war's

Ser_AG: mov S0BUF,#SNo          ; Fehler senden
        sjmp Ser_End            ; warten auf nächsten Befehl

Ser_AE: setb Fat1               ; FatalError 8 setzen (ausnahmsweise 2 Bits!)
        setb Fat4
        ljmp FatalError         ; und weg ist er für immer

Ser_End:pop Acc                 ; A restoren
        pop PSW                 ; PSW restoren
```

```
        reti                    ; das war's

Ser_Kon:cjne A,#MaxCode,Ser_Ko1 ; springen wenn A != MaxCode
        sjmp Ser_Ko2            ; A == MaxCode

Ser_Ko1:jc Ser_Ko2              ; springen wenn A < MaxCode
        sjmp Ser_Ko4            ; Code > MacCode

Ser_Ko2:jnz Ser_Ko3             ; springe wenn Code != 0
Ser_Ko4:clr C                   ; A > MaxCode oder A == 0, Carry löschen
        ret                     ; zurück

Ser_Ko3:setb C                  ; A <= MaxCode und A != 0, Carry setzen
        ret                     ; zurück

Ser_Reg:movc A,@A+PC            ; in Tabelle nachsehen (Code beginnt mit 1!)
        ret                     ; zurück
        DB P1                   ; Tabelle der Variablenadressen
        DB P3
        DB P4
        DB P5
        DB Latch0
        DB Latch1
        DB Modus
        DB mout
        DB msel
        DB sout
        DB atten
        DB Error
        DB LError
        DB Misc
        DB Prell
        DB ctime
        DB dtime
        DB otime
        DB atime
        DB ktime
        DB stime
        DB ABatt
```

Diese Routine ist die größte und komplizierteste Interruptroutine, da sie jegliche Kommunikation über die serielle Schnittstelle verwalten muß. Zunächst wird der Grund für den Interrupt (Transmit oder Receive) ermittelt. Im Fall eines Sendens wird garnichts getan.

Bei einem Empfang unterscheidet die Routine 3 Fälle:

Initialisierung: Beim Start des Multiswitches wird ein Verbindungsaufbau mit einem externen Gerät versucht. Für ca. eine halbe Sekunde wird auf eine Antwort gewartet. Dieser Teil handhabt den aktiven Verbindungsaufbau und teilt das der Initialisierungsroutine mit.

Telefonschnittstelle: Prinzipiell werden von der Telefonschnittstelle nur zwei Mitteilungen erwartet; alles andere wird ignoriert. Erstens kann es eine Antwort auf eine Notfallmitteilung sein, die vermerkt wird, oder es handelt sich um eine Aufforderung zum Verbindungsneuaufbau, der entsprechend behandelt wird.

PC–Modus: In diesem Fall werden die Befehle des PCs empfangen, ausgewertet und befolgt. Dabei wird das Protokoll, wie es im Kapitel 8.3.3 definiert wurde, genau eingehalten. Um die verschiedenen Befehlsteile an der richtigen Reihenfolge auseinanderzukennen, habe ich eine Stufenvariable eingeführt, in der der Fortschritt eines Befehlsablaufes gespeichert wird; weiters wird in einer anderenVariable der aktuelle

Befehl abgespeichert, da beim Eintreffen des Argumentes die Information, welcher Befehl überhaupt abgearbeitet wird, vorhanden sein muß.

9.7.2.3 Initialisierung

```
 ; Inititialisierung
 ; =================

Init:   mov PSW,#0              ; alles zurücksetzen
        mov SP,#Stack           ; Stack setzen
        mov P0,#0               ; für Powerdown
        mov P1,#0FFh            ; P1 = Eingang
        mov P2,#0               ; für Powerdown
        mov P3,#11011111b       ; P3 = Eingang, Ausgang
        mov P4,#00111111b       ; P4 = Eingang, Ausgang
        mov A,RamOk             ; RAM-Testwert holen
        cjne A,#ROK,ICold       ; springe wenn der Testwert nicht ok
        jnb E1,IWarm            ; springe wenn E1 nicht gedrückt
        jnb E2,IWarm            ; springe wenn E2 nicht gedrückt
ICold:  mov RamOK,#ROK          ; RAM-Testwert setzen
        mov Prell,#KPrell       ; Prellzeit auf Default
        mov ctime,#Kctime       ; ctime auf Default
        mov dtime,#Kdtime       ; dtime auf Default
        mov otime,#Kotime       ; otime auf Default
        mov atime,#Katime       ; atime auf Default
        mov ktime,#Kktime       ; ktime auf Default
        mov stime,#Kstime       ; stime auf Default
        mov ABatt,#0            ; aktueller Batteriewert unbekannt
IWarm:  mov Modus,#M_New        ; kein Modus
        mov mout,#links         ; linke Mouse-Taste aktivieren
        mov msel,#click         ; click setzen
        mov sout,#G1            ; Gerät 1 aktivieren
        mov atten,#A_Mod        ; Modus ermitteln
        clr abusy               ; derzeit keine Attentionbehandlung
        mov Latch0,#0FFh        ; alle LEDs aus (Bestimmung später)
        acall mL0               ; Latch 0 aktualisieren
        mov Latch1,#10001111b   ; alle LEDs und Ausgänge aus (Bestimmung später)
        acall mL1               ; Latch 1 aktualisieren
        mov LError,Error        ; alten Fehler sichern
        mov Error,#0            ; kein Fehler
        mov Misc,#0             ; alles zurücksetzen
        mov SerB,#0             ; alles zurücksetzen
        setb STFree             ; Serielle frei!
        clr STel                ; kein Telefon angemeldet
        clr SPC                 ; kein PC angemeldet
        mov Stufe,#Stufe0       ; auf Grundstufe setzen
        mov Kenn,#0             ; keine Kennung
        mov Befehl,#0           ; kein Befehl
        mov ftime,#flash        ; Blinkzeit initialisieren
        mov btime,#Battest      ; Batterietestzeit initialisieren
        mov stimem,stime        ; Zeizzählung für's Scannen initialisieren
        setb STFree             ; Serielle frei
        mov A,P4                ; Port 4 in A zur Weiterbearbeitung
        anl A,#00101010b        ; entscheidende Bits herausschneiden
        mov Port5,P5            ; Port 5 aktualisieren
        mov C,HMin              ; HMin auch ins A
        mov Acc.7,C
        mov TimI,A              ; sichern
        mov C,HM                ; HM sichern
        mov oHM,C
        mov C,ScOn              ; ScOn sichern
        mov oScOn,C
        mov ADCON,#00000110b    ; ADC-Grundeinstellung
        mov TMOD,#00100001b     ; Timer (0 & 1)-Grundeinstellung
        mov SOCON,#01010000b    ; Serielle-Grundeinstellung
        mov PCON,#0             ; kleinere Baudrate und zurücksetzen der Userbits
        mov TH1,#0FDh           ; Reloadvalue für 9600 bzw. 19200 baud
        mov IP0,#00000110b      ; Interruptprioritäten
        mov IP1,#0              ; Timer 2 alle auf 0
        mov IEN1,#0             ; kein Timer2 Interrupt
        mov TL0,#207            ; Lowbyte für Zeiteinstellung
        mov TH0,#213            ; Highbyte für Zeiteinstellung
        mov TCON,#01010101b     ; Timer starten und externe Interrupts edge-triggered

        clr TI                  ; zur Sicherheit
        mov S0Buf,#SEsc         ; SEsc seriell Senden
In_A:   jnb TI,In_A             ; Warten auf Beendigung der Übertragung
```

```
        clr TI                  ; Interruptbit löschen
        mov S0Buf,#SG_MS        ; eigene Kennung senden

        mov IEN0,#10010111b     ; Interrupts erlauben

        setb SInit              ; Kennung für Initialisierung setzen
        mov wait,#IniWait       ; Wartezeit setzen
        acall Warte             ; Warten
        clr SInit               ; Initialisierung vorbei

        ljmp att_tst            ; und los geht's
```

Zunächst werden die grundlegenden Hardwareeinstellungen vorgenommen. Danach wird entschieden, ob alle Variable zurückgesetzt werden sollen (Coldreset) oder nur die Betriebsvariable (Warmreset). Weiters werden alle notwendigen Variable auf ihre Defaultwerte gesetzt, die Hardware initialisiert, die Eingänge abgefragt und gespeichert und zuletzt beide Timer gestartet. Bevor alle Interrupts erlaubt werden, wird für ca. eine halbe Sekunde versucht eine Verbindung mit einem externen seriellen Gerät herzustellen. Zuletzt wird in die `att_tst`–Routine gesprungen um einen Modus zu ermitteln und ins richtige Modul zu springen.

9.7.2.4 Basisunterroutinen

9.7.2.4.1 Comp

```
 ; Basis-Unterroutinen
 ; ====================

 ; Größenvergleich              ; Bit Comp (R0,R1) [Register aus Bank 0!]
 ; ***************

; Setzt C, wenn R0 kleiner oder gleich als R1 (unsigned)
; Löscht C sonst

; Verändert nur C

Comp:   push Acc                ; A sichern
        mov A,R0                ; R0 in A (wegen Adressierungsmoden für cjne)
        cjne A,R1_0,Comp1       ; springen wenn R0 und R1 verschieden
        sjmp Comp_S             ; R0 == R1

Comp1:  jc Comp_S               ; springen wenn R0<R1
        pop Acc                 ; A wieder zurück
        clr C                   ; R0>R1, Carry löschen
        ret                     ; zurück

Comp_S: pop Acc                 ; A wieder zurück
        setb C                  ; R0<=R1, Carry setzen
        ret                     ; zurück
```

Diese Routine vergleicht die vorzeichenlosen Werte in den Registern 0 und 1 in der Bank 0 und setzt das Carrybit entsprechend. Diese Funktion wird für die diversen Zeitmessungen benötigt.

9.7.2.5 sonstige Unterroutinen

9.7.2.5.1 CompTA

```
 ; Unterroutinen
 ; =============

 ; CompTA                      ; Bit CompTA (void)
 ; ******                      ; setzt Carry wenn ticks<=atime

CompTA:                        ; Vergleichen von ticks und atime
        mov R0,ticks           ; ticks in R0 zum Vergleich
        mov R1,atime           ; atime in R1 zum Vergleich
        acall Comp             ; Vergleich
        ret                    ; das war's
```

Diese Funktion vergleicht die aktuell vergangenen ticks mit der `atime` und setzt das Carrybit als Antwort.

9.7.2.5.2 mL1

```
 ; mL1                        ; void mL1 (void)
 ; ***                        ; schreibt den aktuellen Wert von Latch1 ins Latch 1

mL1:    mov DPTR,#XL1          ; Adresse (#) des Latches 1 laden
        mov A,Latch1           ; Wert des Latches 1 laden
        movx @DPTR,A           ; ins Latch 1 speichern
        ret
```

Die Variable Latch1 wird ins externe Latch 1 kopiert.

9.7.2.5.3 mL0

```
 ; mL0                        ; void mL0 (void)
 ; ***                        ; schreibt den aktuellen Wert von Latch0 ins Latch 0

mL0:    mov DPTR,#XL0          ; Adresse des Latches 0 laden
        mov A,Latch0           ; Wert des Latches 0 laden
        movx @DPTR,A           ; ins Latch 0 speichern
        ret
```

Die Variable Latch0 wird ins externe Latch 0 kopiert.

9.7.2.5.4 LEDsOn

```
 ; LEDsOn                     ; void LEDsOn (void)
 ; ******                     ; schaltet alle LEDs ein

; Verändert Latch0 & Latch1

LEDsOn: mov LL0,Latch0         ; Latch0 sichern
        clr A                  ; A löschen
        mov DPTR,#XL0          ; Adresse des Latches 0 laden
        movx @DPTR,A           ; Latch 0 löschen (alle LEDs ein)
        mov Latch0,A           ; neuen Wert ins Latch0
        mov LL1,Latch1         ; Latch1 sichern
        mov A,Latch1           ; Latch 1 ins A für Transfer
        anl A,#11110000b       ; LEDs ein
        mov Latch1,A           ; neuen Wert ins Latch1
        mov DPTR,#XL1          ; Adresse des Latches 1 laden
        movx @DPTR,A           ; ins Latch 1 speichern
        ret
```

Die aktuellen Einstellungen der LEDs werden gespeichert und alle LEDs werden eingeschaltet.

9.7.2.5.5 LEDsOff

```
 ; LEDsOff                     ; void LEDsOff (void)
 ; *******                     ; restored die Latches wieder

; Verändert Latch0 & Latch1

LEDsOff:mov Latch0,LL0         ; Latch0 restoren
        acall mL0              ; Latch 0 restaurieren
        mov Latch1,LL1         ; Latch1 restoren
        acall mL1              ; Latch 1 restaurieren
        ret
```

Die „alten“ LED–Einstellungen werden wieder aktiviert.

9.7.2.5.6 StartInt

```
 ; StartInt                    ; void StartInt (void)
 ; ********                    ; erlaubt alle notwendigen Interrupts

StartInt:
        mov IEN0,#10010111b    ; global, UART, Ex1, Timer 0, Ex0 ein
        ret                    ; das war's
```

Alle Interrupts werden erlaubt.

9.7.2.5.7 StopInt

```
 ; StopInt                     ; void StopInt (void)
 ; *******                     ; verbietet fast alle Interrupts

StopInt:
        mov IEN0,#10000110b    ; global, Ex1 ein
        ret                    ; das war's
```

Die Interrupts werden auf den Timer0– und externen Interrupt 1 eingeschränkt.

9.7.2.5.8 OWarte

```
 ; OWarte                      ; void OWarte (void)
 ; ******                      ; Wartet otime lange

OWarte: mov wait,otime         ; otime in die Wartevariable
        acall Warte            ; Warten
        ret                    ; das war's
```

Es wird `otime` lange gewartet. Diese Routine ist für die Ausgangsimpulse wichtig.

9.7.2.5.9 Entprell

```
 ; Entprell                    ; void Entprell (void)
 ; ********                    ; Warten bis der Taster ausgeprellt hat

Entprell:
        mov wait,Prell         ; Setzen der Entprellzeit
        acall Warte            ; in die eigentliche Warteroutine springen
        ret                    ; geschafft
```

Es wird die Entprellzeit `Prell` abgewartet.

9.7.2.5.10 Warte

```
 ; Warte                        ; void Warte (wait) [wait wird nur von Interrupt verwendet]
 ; *****

Warte:                          ; eigentliche Warteroutine
       clr wready               ; Löschen des Wartebits
Wart_A: jnb wready,Wart_A       ; wready wird von Interrutroutine gesetzt wenn es so weit ist
       ret                      ; das war's
```

Dies ist die eigentliche Warteroutine, die von den obigen Routinen (z.B. `Entprell`, `OWarte`, ...) aufgerufen wird.

9.7.2.5.11 Puls

```
 ; Puls                         ; void Puls (void)
 ; ****

Puls:                           ; aktiviert den zuständigen Ausgang in seiner spezifischen Art
       jnb MMon,Pu_A            ; springe wenn MouseMode = off
       cjne mselr,#click,Pu_B   ; springe wenn mousesel != click
       acall Clickit            ; einen Click an den aktiven Ausgang
       ret                      ; das war's

Pu_A:  ajmp Pu_AA               ; Verlängerung des relativen Sprunges

Pu_B:  cjne mselr,#dclick,Pu_C  ; springe wenn mousesel != dclick
       acall Clickit            ; einen Click an den aktiven Ausgang
       cjne moutr,#M_Off,Pu_D   ; springe wenn mouseout != MouseOff
       ret                      ; das war's

Pu_D:  mov wait,dtime           ; Doppelclickzeit ins "Warteregister"
       acall Warte              ; Abwarten der Pause
       acall Clickit            ; zweiten Click an den aktiven Ausgang
       ret                      ; das war's

Pu_C:  cjne mselr,#switch,Pu_E  ; springe wenn mousesel != switch
       cjne moutr,#M_Off,Pu_F   ; springe wenn mouseout != M_Off
       acall Clickit            ; Click an den aktiven Ausgang
       ret                      ; das war's

Pu_F:  cjne moutr,#rechts,Pu_G  ; springe wenn mouseout != rechts
       cpl MRT                  ; Komplement der rechten Mousetaste
       sjmp Pu_H                ; zum nächsten Sammelpunkt

Pu_G:  cjne moutr,#mitte,Pu_I   ; springe wenn mouseout != mitte
       cpl MMT                  ; Komplement der mittigen Mousetaste
       sjmp Pu_H                ; zum nächsten Sammelpunkt

Pu_I:  cjne moutr,#links,Pu_J   ; springe wenn mouseout != links
       cpl MLT                  ; Komplement der linken Mousetaste
       sjmp Pu_H                ; zum nächsten Sammelpunkt

Pu_J:  setb Fat5                ; Fatal 5 Error setzen
       ljmp FatalError          ; und niemals kehrt er wieder

Pu_H:  acall mL1                ; Latch 1 aktualisieren
       ret

Pu_E:  cjne mselr,#nextm,Pu_J2  ; springe wenn mousesel != nextm
       cjne moutr,#links,Pu_K   ; springe wenn mouseout != links
       setb MLSel               ; LED für linke Mousetaste off
       jb MoMOn,Pu_L            ; springe wenn MouseMitte off
       mov mout,#mitte          ; mittlere Mousetaste als aktuell setzen
       clr MMSel                ; LED für mittlere Mousetaste on
       sjmp Pu_M                ; zum nächsten Sammelpunkt

Pu_J2: setb Fat2                ; Fatal 2 Error setzen
       ljmp FatalError          ; und niemehr kehrte er wieder

Pu_L:  mov mout,#rechts         ; rechte Mousetaste als aktuell setzen
       clr MRSel                ; LED für rechte Mousetaste on
       sjmp Pu_M                ; zum nächsten Sammelpunkt

Pu_K:  cjne moutr,#mitte,Pu_N   ; springe wenn mouseout != mitte
       setb MMSel               ; LED für mittlere Mousetaste off
```

```
        sjmp Pu_L               ; rechte Taste aktivieren

Pu_N:   cjne moutr,#rechts,Pu_O ; springe wenn mouseout != rechts
        setb MRsel              ; LED für rechte Mousetaste off
        mov Port5,P5            ; "internen" Port5 aktualisieren
        jnb HMin,Pu_P           ; springe wenn Hardwaremouseswitch in
        mov mout,#M_off         ; MOff Option als aktuell setzen
        clr MOff                ; LED für MOff on
        sjmp Pu_M               ; zum nächsten Sammelpunkt

Pu_P:   mov mout,#links         ; linke Mousetaste als aktuell setzen
        clr MLSel               ; LED für linke Mousetaste on
        sjmp Pu_M               ; zum nächsten Sammelpunkt

Pu_O:   cjne moutr,#M_Off,Pu_J  ; springe wenn mouseout != M_Off
        setb MOff               ; LED für MOff off
        sjmp Pu_P               ; linke Taste aktivieren

Pu_M:   acall mL0               ; Latch 0 aktualisieren
        ret                     ; das wars

Pu_AA:  cjne soutr,#G1,Pu_AB    ; springe wenn sout != G1
        setb A1                 ; A1 on
        acall OWarte            ; otime abwarten
        clr A1                  ; A1 off
        ret                     ; das war's

Pu_AB:  cjne soutr,#G2,Pu_AC    ; springe wenn sout != G2
        setb A2                 ; A2 on
        acall OWarte            ; otime abwarten
        clr A2                  ; A2 off
        ret                     ; das war's

Pu_AC:  cjne soutr,#G3,Pu_AD    ; springe wenn sout != G3
        cpl A3                  ; A3 invertieren
        ret                     ; das war's

Pu_AD:  cjne soutr,#M_on,Pu_AE  ; springe wenn sout != M_on
        mov atten,#A_SWSM       ; Attention setzen
        ret                     ; das war's

Pu_AE:  setb Fat1               ; Fatal Error 1 setzen
        ljmp FatalError         ; und niemals kehrte er wieder zurück
```

Zunächst wird zwischen Mousemode und Switchmode unterschieden und in die entsprechenden Teile der Routine verzweigt. Danach wird im Mousemode nach der Bedienungsart unterschieden und jeder Ausgang entsprechend aktiviert. Im Switchmode wird nach den verschiedenen Ausgängen unterschieden und diese werden dementsprechend aktiviert.

9.7.2.5.12 Clickit

```
 ; Clickit                     ; void Clickit (void)
 ; *******

Clickit:                        ; aktiviert den entsprechenden Mouseausgang für ctime
        cjne moutr,#links,Cli_A ; springen wenn mout != links
        setb MLT                ; linke Mousetaste on
        acall mL1               ; Latch 1 aktualisieren
        acall WClick            ; Warten
        clr MLT                 ; linke Mousetaste off
        acall mL1               ; Latch 1 aktualisieren
        ret                     ; das war's

Cli_A:  cjne moutr,#mitte,Cli_B ; springen wenn mout != mitte
        setb MMT                ; mittlere Mousetaste on
        acall mL1               ; Latch 1 aktualisieren
        acall WClick            ; Warten
        clr MMT                 ; mittlere Mousetaste off
        acall mL1               ; Latch 1 aktualisieren
        ret                     ; das war's

Cli_B:  cjne moutr,#rechts,Cli_C; springen wenn mout != rechts
        setb MRT                ; rechte Mousetaste on
        acall mL1               ; Latch 1 aktualisieren
```

```
        acall WClick            ; Warten
        clr MRT                 ; rechte Maustaste off
        acall mL1               ; Latch 1 aktualisieren
        ret                     ; das war's

Cli_C:  cjne moutr,#M_off,Cli_D ; springen wenn mout != M_off
        mov atten,#A_SWMS       ; Moduswechsel beantragen
        ret                     ; das war's

Cli_D:  setb Fat5               ; Fatal Error 5 setzen
        ljmp FatalError         ; deadend
```

Diese Routine wird von `Puls` aufgerufen und aktiviert die gerade aktive Mousetaste für die Zeit `ctime`.

9.7.2.5.13 WClick

```
 ; WClick                      ; Wvoid Click (void)
 ; ******

WClick:                       ; wartet ctime lange
        mov wait,ctime          ; ctime in wait für Warten
        acall Warte             ; Warten
        ret                     ; das war's
```

Wartet für die Funktion `Clickit` die Zeit `ctime`.

9.7.2.5.14 Nextout

```
 ; Nextout                     ; void Nextout (void)
 ; *******

Nextout:                      ; aktiviert den nächsten erlaubten Ausgang
        jb MMon,Nextmouse       ; springe wenn im Mousemode
        cjne soutr,#G1,NexO_A   ; springe wenn sout != Gerät 1
        setb A1Sel              ; LED für A1 off
        jb out2On,NexO_B        ; springe wenn Gerät 2 deaktiviert
        mov sout,#G2            ; Gerät 2 setzen
        clr A2Sel               ; LED für A2 on
        sjmp NexO_G             ; nächster Sammelpunkt

NexO_B: jb out3On,NexO_C        ; springe wenn Gerät 3 deaktiviert
        mov sout,#G3            ; Gerät 3 setzen
        clr A3Sel               ; LED für A3 on
        sjmp NexO_G             ; nächster Sammelpunkt

NexO_C: mov Port5,P5            ; Port 5 aktualisieren
        jnb HMin,NexO_H         ; springe wenn Hardwaremouse in
        jb MoOn,NexO_H          ; springe wenn Mousemode per DIP off
        mov sout,#M_on          ; Mousemodeaktivierung setzen
        clr MOn                 ; LED für MOn on
        sjmp NexO_G             ; nächster Sammelpunkt

NexO_H: clr A1Sel               ; LED für A1 reaktivieren
        sjmp NexO_G             ; nur Gerät 1 aktivierbar!

NexO_A: cjne soutr,#G2,NexO_D   ; springe wenn sout != Gerät 2
        setb A2Sel              ; LED für A2 off
        jb out3On,NexO_E1       ; springe wenn Gerät 3 deaktiviert
        mov sout,#G3            ; Gerät 3 setzen
        clr A3Sel               ; LED für A3 on
        sjmp NexO_G             ; nächster Sammelpunkt

NexO_E: mov sout,#G1            ; Gerät 1 setzen
        clr A1Sel               ; LED für Gerät on
        sjmp NexO_G             ; nächster Sammelpunkt

NexO_D: cjne soutr,#G3,NexO_F   ; springe wenn sout != Gerät 3
        setb A3Sel              ; LED für A3 off
NexO_E1:mov Port5,P5            ; Port5 aktualisieren
        jnb HMin,NexO_E         ; springe wenn HardwareMouse in
        jb MoOn,NexO_E          ; springe wenn MouseMode per DIP off
        mov sout,#M_on          ; Mousemodeaktivierung setzen
```

```
        clr MOn                 ; LED für MOn on
        sjmp NexO_G             ; nächster Sammelpunkt

NexO_F: cjne soutr,#M_On,NexO_I ; springe wenn sout != MouseOn
        setb MOn                ; LED für MouseOn off
        sjmp NexO_E             ; Gerät 1 aktivieren

NexO_I: setb Fat1               ; Fatal Error 1 setzen
        ljmp FatalError         ; never come back

NexO_G: acall mL1               ; Latch 1 aktualisieren
        ret                     ; das war's
```

Diese Funktion unterscheidet zu Beginn zwischen Mousemode und Switchmode und springt im Falle des Mousemodes in die Funktion `Nextmouse`. Im Falle des Switchmodes wird der nächste gültige Ausgang aktiviert, wobei berücksichtig wird, daß die Ausgänge 2 und 3 per DIP–Schalter abgeschaltet werden können und daß der Mousemode nur dann per Software aktiviert werden darf, wenn kein Hardware–Mouse–Schalter eingesteckt ist.

9.7.2.5.15 Nextmouse

```
 ; Nextmouse                  ; void Nextmouse (void)
 ; *********

Nextmouse:                      ; wählt den nächsten Mousetastenmodus (click, ...)
        cjne mselr,#click,NeM_A ; springe wenn  msel != click
        setb MC                 ; LED für Mouse Click off
        mov msel,#dclick        ; dclick als aktuell setzen
        clr MDC                 ; LED für Mouse DClick on
        sjmp NeM_E              ; zum nächsten Sammelpunkt

NeM_A:  cjne mselr,#dclick,NeM_B; springe wenn msel != dclick
        setb MDC                ; LED für Mouse DClick off
        mov msel,#switch        ; switch als aktuell setzen
        clr MS                  ; LED für Mouse Switch on
        sjmp NeM_E              ; zum nächsten Sammelpunkt

NeM_B:  cjne mselr,#switch,NeM_C; springe wenn msel != switch
        setb MS                 ; LED für Mouse Switch off
        mov msel,#nextm         ; nextm als aktuell setzen
        clr Next                ; LED für Mouse Next on
        sjmp NeM_E              ; zum nächsten Sammelpunkt

NeM_C:  cjne mselr,#nextm,NeM_D ; springe wenn msel != nextm
        setb Next               ; LED für Mouse Next off
        mov msel,#click         ; Mouse Click als aktuell setzen
        clr MC                  ; LED für Mouse Click on
        sjmp NeM_E              ; zum nächsten Sammelpunkt

NeM_D:  setb Fat2               ; Fatal Error 2 setzen
        ljmp FatalError         ; und niemehr kam er wieder

NeM_E:  acall mL0               ; Latch 0 aktualisieren
        ret                     ; das war's
```

Dise Funktion arbeit genauso wie `Nextout` für den Switchmode, nur daß sie bei aktivem Mousemode verwendet wird.

9.7.2.5.16 Stepselect

```
 ; Stepselect                 ; void Stepselect (void)
 ; **********

StepSelect:                     ; wählt den nächsten Ausgang und wartet auf E3
        acall Nextout           ; nächster Ausgang
Step_A: jb E3,Step_A            ; warten auf Loslassen von E3
        ret                     ; das war's
```

Diese Funktion vereint `Nextout` mit einem Warten auf das Loslassen von E3.

9.7.2.5.17 FatalError

```
 ; Notfallende
 ; ===========

FatalError:                     ; stellt alles auf aus und geht in den Schlafmodus
        clr EA                  ; alle Interrupts aus
        mov Latch0,#0FFh        ; alles aus im Latch 0
        acall mL0               ; Latch 0 aktualisieren
        mov Latch1,#10001111b   ; alles aus im Latch 1
        acall mL1               ; Latch 1 aktualisieren
        mov P0,#0               ; alle Ports löschen
        mov P1,#0
        mov P2,#0
        mov P3,#0
        mov P4,#0
Fatal:  mov PCON,#00000010b     ; in den PowerDownModus
        sjmp Fatal              ; nur zur Sicherheit
```

Diese Funktion wird bei einem schweren internen Fehler aufgerufen und soll verhindern, daß das außer Kontrolle geratene Programm Schaden anrichtet. Daher wird der Prozessor in den Power–Down–Mode geschaltet.

9.7.3 die Module

9.7.3.1 Mod1

```
 ; Module
 ; ======

 ; Modus 1
 ; *******

Mod1:   cjne att,#A_OK,atst1    ; Springe wenn Moduswechsel beantragt
        jnb E1,Mod1             ; wenn E1=off dann warte
        mov ticks,#0            ; ticks=0 zur Zeitmessung
        mov ticks2,#0           ; ticks2=0 zur Zählung der atime
        acall Entprell          ; Entprellen
Mo1_D:  acall CompTA            ; Vergleich von ticks und atime
        jnc Mo1_C               ; springe wenn ticks>atime
        jb E1,Mo1_D             ; ticks<=atime, springe wenn E1 immer noch aktiv
        acall Puls              ; OK, Aktivierung
        sjmp Mod1               ; und von neuem

Mo1_C:  jb E1,Mo1_E             ; springe wenn immer noch aktiv
        acall Nextout           ; nächster Ausgang
        acall Entprell          ; Entprellen
        sjmp Mod1               ; und von neuem

Mo1_E:  cjne att,#A_Mod,Mo1_F   ; springe wenn kein Moduswechel beantragt ist
        ljmp att_tst            ; ermittle neuen Modus

Mo1_F:  mov R0,atime            ; atime in R0 für Vergleich
        mov R1,ticks2           ; ticks2 in R1 für Vergleich
        acall Comp              ; Vergleich
        jc Mo1_G                ; springe wenn atime<=ticks2
        sjmp Mo1_D              ; noch nicht

Mo1_G:  mov ticks2,#0           ; Zurücksetzen von ticks2
        acall Nextout           ; nächster Ausgang
Mo1_H:  mov R0,ktime            ; ktime in R0 für Vergleich
        mov R1,ticks2           ; ticks2 in R1 für Vergleich
        acall Comp              ; Vergleich
        jnc Mo1_I               ; springe wenn ticks2<=ktime
        mov ticks2,#0           ; Zurücksetzen von ticks2
        sjmp Mo1_C              ; noch nicht aus

Mo1_I:  jb E1,Mo1_H             ; springen wenn E1 immer noch on
        sjmp Mod1               ; alles von vorne

 ; Sprunghilfe 1
 ; *************

atst1:  ljmp att_tst            ; zur Verlängerung der relativen Sprünge
```

Das Modul für den Modus 1 setzt die Definitionen aus dem Kapitel 4.1.1 genau um. Bei jedem Durchlauf wird auf eine vorhandene Message geprüft. Sollte eine Message vorhanden sein, so wird die Programmkontrolle an `att_tst` übergeben, die diese auswertet. Die Sprunghilfe mußte eingeführt werden, da mit einem relativen Sprung `att_tst` nicht erreicht werden konnte.

9.7.3.2 Mod2

```
 ; Modus 2
 ; *******

Mod2:   cjne att,#A_OK,atst1    ; Springe wenn Moduswechsel beantragt
        jnb E1,Mo2_A            ; springe wenn E1 nicht gedrückt
        jb MMon,Mo2_N           ; springe wenn im Mousemode
        mov ticks,#0            ; Zurücksetzen von ticks
        cjne soutr,#G3,Mo2_B    ; springe wenn sout != 3
Mo2_N:  acall Puls              ; Ausgang ändern
        acall Entprell          ; Entprellen
Mo2_C:  jb E1,Mo2_C             ; warten auf E1 loslassen
Mo2_D:  jb E3,Mo2_D             ; warten auf E3 loslassen
        sjmp Mod2               ; und alles von vorne

Mo2_B:  cjne soutr,#M_On,Mo2_B0 ; springe wenn sout != M_On
        sjmp Mo2_N              ; nur aktivieren

Mo2_B0: cjne soutr,#G2,Mo2_E    ; springe wenn sout != 2
        setb A2                 ; Ausgang 2 aktivieren
        sjmp Mo2_F              ; zum nächsten Sammelpunkt

Mo2_E:  cjne soutr,#G1,Mo2_G    ; springe wenn sout != 1
        setb A1                 ; Ausgang 1 aktivieren
        sjmp Mo2_F              ; zum nächsten Sammelpunkt

Mo2_G:  setb Fat1               ; Fatal1-Error setzen
        ljmp FatalError         ; zur Fehlerbehandlung

Mo2_F:  acall Entprell          ; Entprellen
Mo2_H:  acall CompTA            ; Vergleich von ticks und atime
        jnc Mo2_I               ; springe wenn nicht ticks<=atime
        jb E1,Mo2_H             ; auf loslassen werten
Mo2_J:  mov R0,ticks            ; ticks in R0 für den Vergleich
        mov R1,otime            ; otime in R1 für den Vergleich
        acall Comp              ; Vergleich
        jc Mo2_J                ; auf Ablauf der Zeit warten
        sjmp Mo2_K              ; zum nächsten Sammelpunkt

Mo2_I:  jb E1,Mo2_I             ; warten auf loslassen
Mo2_K:  cjne soutr,#G1,Mo2_L    ; springen wenn sout != 1
        clr A1                  ; Ausgang 1 aus
        sjmp Mod2               ; alles von vorne

Mo2_L:  cjne soutr,#G2,Mo2_M    ; springen wenn sout != 2
        clr A2                  ; Ausgang 2 aus
        sjmp Mod2               ; alles von vorne

Mo2_M:  setb Fat1               ; Fatal1-Error setzen
        ljmp FatalError         ; zur Fehlerbehandlung

Mo2_A:  jnb E3,Mod2             ; springe, wenn auch E3 nicht gedrückt
        acall Entprell          ; E3 entprellen
        acall Stepselect        ; nächster Ausgang + E3-Block
        sjmp Mod2               ; alles von vorne
```

Das Modul für den Modus 2 setzt die Definitionen aus dem Kapitel 4.1.2 genau um. Bei jedem Durchlauf wird auf eine vorhandene Message geprüft. Sollte eine Message vorhanden sein, so wird die Programmkontrolle an `att_tst` übergeben, die diese auswertet.

9.7.3.3 Mod3

```
 ; Modus 3
 ; *******

Mod3:   cjne att,#A_OK,atst2    ; springe wenn Moduswechsel beantragt
        jnb E2,Mod3             ; wenn E2=off dann warte
        mov ticks,#0            ; ticks zurücksetzen
        acall Entprell          ; Entprellen
Mo3_A:  jnb E2,Mo3_B            ; springe wenn E2 schon losgelassen
        acall CompTA            ; Vergleich von ticks und atime
        jc Mo3_A                ; noch innerhalb der Wartezeit
        setb Bed1               ; Bedienungsfehler 1 aktivieren
Mo3_C:  jnb E2,Mo3_D            ; springe wenn E2 endlich wieder deaktiviert
        cjne att,#A_Mod,Mo3_C   ; springe wenn E2 nicht abgesteckt
        clr Bed1                ; Bedienungsfehler 1 wieder löschen
        sjmp atst2              ; Modus überprüfen

Mo3_D:  clr Bed1                ; Bedienungsfehler 1 wieder löschen
        sjmp Mod3               ; alles wieder von vorne

Mo3_B:  jnb E2,Mo3_E            ; springe wenn E2 nicht mehr aktiv
        acall Entprell          ; Entprellen
Mo3_F:  jnb E2,Mo3_G            ; springen wenn E2 nicht mehr aktiv
        acall CompTA            ; Vergleich von ticks und atime
        jc Mo3_F                ; springen wenn noch Zeit
        setb Bed2               ; Bedienungsfehler 2 setzen
Mo3_H:  jnb E2,Mo3_I            ; springen wenn E2 endlich deaktiviert
        cjne att,#A_Mod,Mo3_H   ; springen wenn E2 nicht abgesteckt
        clr Bed2                ; Bedienungsfehler 2 wieder löschen
        sjmp atst2              ; Modus überprüfen

Mo3_I:  clr Bed2                ; Bedienungsfehler 2 wieder löschen
        sjmp Mod3               ; wieder alles von vorne

Mo3_G:  acall Nextout           ; nächster Ausgang
        sjmp Mod3               ; alles wieder von vorne

Mo3_E:  acall CompTA            ; Vergleich von ticks und atime
        jc Mo3_B                ; springen wenn noch Zeit
        acall Puls              ; Ausgang aktivieren
        sjmp Mod3               ; alles wieder von vorne

 ; Sprunghilfe 2
 ; *************

atst2:  ljmp att_tst            ; zur Verlängerung der relativen Sprünge
```

Das Modul für den Modus 3 setzt die Definitionen aus dem Kapitel 4.1.3 genau um. Bei jedem Durchlauf wird auf eine vorhandene Message geprüft. Sollte eine Message vorhanden sein, so wird die Programmkontrolle an `att_tst` übergeben, die diese auswertet. Die Sprunghilfe mußte eingeführt werden, da mit einem relativen Sprung `att_tst` nicht erreicht werden konnte.

9.7.3.4 Mod4

```
 ; Modus 4
 ; *******

Mod4:   cjne att,#A_OK,atst2    ; springe wenn Moduswechsel beantragt
        jnb E2,Mo4_A            ; springe wenn E2 nicht gedrückt
        mov ticks,#0            ; ticks zurücksetzen
        acall Entprell          ; Entprellen
Mo4_C:  jnb E2,Mo4_D            ; springen wenn schon losgelassen
        acall CompTA            ; Vergleich von ticks und atime
        jc Mo4_C                ; springe wenn noch in der Zeit
        setb Bed1               ; Bedienungsfehler 1 setzen
Mo4_E:  jnb E2,Mo4_F            ; springe wenn endlich losgelassen
        cjne att,#A_Mod,Mo4_E   ; springe wenn E2 nicht abgesteckt
        clr Bed1                ; Bedienungsfehler 1 löschen
        sjmp atst2              ; überprüfen des Moduses
```

```
Mo4_F:  clr Bed1                ; Bedienungsfehler 1 löschen
        sjmp Mod4               ; alles wieder von vorne

Mo4_D:  jnb E2, Mo4_G           ; springen wenn E2 noch nicht gedrückt
        lcall Entprell          ; Entprellen
Mo4_H:  jnb E2,Mo4_I            ; springe wenn E2 wieder losgelassen
        lcall CompTA            ; Vergleich von ticks und atime
        jc Mo4_H                ; springe wenn noch genug Zeit
        setb Bed2               ; Bedienungsfehler 2 setzen
Mo4_L:  jnb E2,Mo4_K            ; springe wenn E2 endlich losgelassen
        cjne att,#A_Mod,Mo4_L   ; springe wenn E2 nicht abgesteckt
        clr Bed2                ; Bedienungsfehler 2 wieder löschen
        sjmp atst2              ; Modus überprüfen

Mo4_K:  clr Bed2                ; Bedienungsfehler 2 wieder löschen
        sjmp Mod4               ; alles wieder von vorne

Mo4_I:  lcall Puls              ; Ausgang aktivieren
        sjmp Mod4               ; alles wieder von vorne

Mo4_G:  lcall CompTA            ; Vergleich von ticks und atime
        jc Mo4_D                ; springe wenn noch genug Zeit
Mo4_J:  jb E3,Mo4_J             ; warten auf Loslassen von E3
        sjmp Mod4               ; alles wieder von vorne

Mo4_A:  jnb E3,Mod4             ; gar nichts gedrückt
        lcall Entprell          ; E3 entprellen
        lcall Stepselect        ; nächster Ausgang + E3-Block
        sjmp Mod4               ; und alles von vorne
```

Das Modul für den Modus 4 setzt die Definitionen aus dem Kapitel 4.1.4 genau um. Bei jedem Durchlauf wird auf eine vorhandene Message geprüft. Sollte eine Message vorhanden sein, so wird die Programmkontrolle an `att_tst` übergeben, die diese auswertet.

9.7.3.5 Mod5

```
 ; Modus 5
 ; *******

Mod5:   cjne att,#A_OK,atst3    ; springe wenn Moduswechsel beantragt
        sjmp Mod5               ; alles andere in der Interruptroutine
```

Das Modul für den Modus 5 prüft andauernd auf eine mögliche Message, da alle anderen Funktionen direkt von der Interruptroutine erledigt werden. Sollte eine Message vorhanden sein, so wird die Programmkontrolle an `att_tst` übergeben, die diese auswertet.

9.7.3.6 Mod6

```
 ; Modus 6
 ; *******

Mod6:   cjne att,#A_OK,atst3    ; springe wenn Moduswechsel beantragt
        jnb E1,Mod6             ; warten auf Aktivität
        lcall Entprell          ; Entprellen
        lcall Puls              ; Ausgang aktivieren
Mo6_A:  jb E1,Mo6_A             ; Warten auf Loslassen von E1
        sjmp Mod6               ; und alles von vorne

 ; Sprunghilfe 3
 ; *************

atst3:  ljmp att_tst            ; zur Verlängerung der relativen Sprünge
```

Alle Aktivitäten, außer dem Auslösen eines Clicks oder Impulses, erledigt die Interruptroutine. Da die Impulse oder Clicks (je nach aktivem Modus) von den Unterroutinen

der letzten Schichte ausgeführt werden, ist dieses Modul extrem kurz. Es muß nur, wie die anderen Module auf eine Message warten und entsprechend reagieren, oder eine Betätigung des Sensors E1 registrieren und die wenigen Unterroutinen aufrufen.

9.7.3.7 Mod7

```
 ; Modus 7
 ; *******

Mod7:   cjne att,#A_OK,atst3    ; springe wenn Moduswechsel beantragt
        jnb E2,Mod7             ; warten auf Aktion
        mov ticks,#0            ; ticks zurücksetzen
        lcall Entprell          ; Entprellen
Mo7_A:  jnb E2,Mo7_B            ; springen wenn E2 schon deaktiviert
        lcall CompTA            ; Vergleich von ticks und atime
        jc Mo7_A                ; springe wenn noch innerhalb der Zeit
        setb Bed1               ; Bedienungsfehler 1 setzen
Mo7_C:  jnb E2,Mo7_D            ; springe wenn E2 endlich deaktiviert
        cjne att,#A_Mod,Mo7_C   ; springe wenn Moduswechsel beantragt
        clr Bed1                ; Bedienungsfehler 1 löschen
        sjmp atst3              ; Modus überprüfen

Mo7_D:  clr Bed1                ; Bedienungsfehler 1 löschen
        sjmp Mod7               ; alles wieder von vorne

Mo7_B:  jnb E2,Mo7_E            ; springen wenn E2 noch nicht gedrückt
        lcall Entprell          ; Entprellen
Mo7_F:  jnb E2,Mo7_G            ; springe wenn nicht mehr aktiv
        lcall CompTA            ; Vergleich von ticks und atime
        jc Mo7_F                ; springe wenn noch Zeit
        setb Bed2               ; Bedienungsfehler 2 setzen
Mo7_H:  jnb E2,Mo7_I            ; springe wenn E2 endlich losgelassen
        cjne att,#A_Mod,Mo7_H   ; springe wenn nicht abgesteckt
        clr Bed2                ; Bedienungsfehler 2 löschen
        sjmp atst3              ; Modus überprüfen

Mo7_I:  clr Bed2                ; Bedienungsfehler 2 löschen
        sjmp Mod7               ; alles wieder von vorne

Mo7_G:  lcall Puls              ; aktiven Ausgang aktivieren
        sjmp Mod7               ; alles wieder von vorne

Mo7_E:  lcall CompTA            ; Vergleich von ticks und atime
        jc Mo7_B                ; springe wenn noch genug Zeit
        sjmp Mod7               ; dann eben nicht
```

Diese Routine ist gegenüber dem Modul für den Modus 6 viel länger, da ein Doppelclick erkannt werden muß, was einen — wie man sieht — deutlich höheren Aufwand verursacht. Dabei werden genau die Definitionen aus dem Kapitel 4.2.2 verwirklicht.

9.7.3.8 Mod8

```
 ; Modus 8
 ; *******

Mod8:   cjne att,#A_OK,att_tst  ; springe wenn Moduswechsel beantragt
        sjmp Mod8               ; alles andere in der Interruptroutine
```

Das Modul für den Modus 8 ist derzeit identisch mit dem des Moduses 5. Es wurde aber trotzdem ein eigenes Modul eingeführt, um zukünftige Änderungen schnell und einfach zu ermöglichen.

9.7.4 Messageauswertung

```
 ; Attention Test
 ; **************

att_tst:setb abusy              ; Attentionbearbeitung startet
        cjne att,#A_OK,att_B1   ; springe wenn atten != A_OK (eigentlich immer!)
att_A:  cjne Modr,#M_NEW,att_C_ ; springe wenn Modus != M_New
        jb E3in,att_D           ; springe wenn E3 nicht in
        jb E2in,att_E           ; springe wenn E2 nicht in
        jb E1in,att_F           ; springe wenn E1 nicht in
att_G:  setb Ansch              ; Anschlußfehler setzen
        mov Modus,#M_New        ; noch kein Modus gefunden
        mov atten,#A_Mod        ; Message: neuer Modus!
        sjmp att_tst            ; noch mal das ganze

att_C_: ajmp att_C              ; Sprungverlängerung

att_B1: ajmp att_B              ; Sprungverlängerung

att_F:  clr Ansch               ; E3 in, E2 in, E1 nicht in
        jnb MMon,att_F1         ; springe wenn kein Mousemode
        jb ScOn,att_F1          ; springe wenn kein Scanmode
        mov Modus,#M_7          ; Modus 7 setzen
        mov atten,#A_OK         ; Mitteilung löschen
        acall att_AA            ; LEDs in Ordnung bringen
        clr abusy               ; Attention verlassen
        ljmp Mod7               ; und los geht's

att_F1: mov Modus,#M_4          ; Modus 4 setzen
        mov atten,#A_OK         ; Mitteilung löschen
        acall att_AA            ; LEDs in Ordnung bringen
        clr abusy               ; Attention verlassen
        ljmp Mod4               ; und los geht's

att_E:  jb E1in,att_G           ; E3 in, E2 nicht in | springe wenn E1 nicht in
        clr Ansch               ; E3 in, E2 nicht in, E1 in
        jb MMon,att_E1          ; springe wenn kein Mousemode
        jb ScOn,att_E1          ; springe wenn kein Scanmode
        mov Modus,#M_6          ; Modus 6 setzen
        mov atten,#A_OK         ; Mitteilung löschen
        acall att_AA            ; LEDs in Ordnung bringen
        clr abusy               ; Attention verlassen
        ljmp Mod6               ; und los geht's

att_E1: mov Modus,#M_2          ; Modus 2 setzen
        mov atten,#A_OK         ; Mitteilung löschen
        acall att_AA            ; LEDs in Ordnung bringen
        clr abusy               ; Attention verlassen
        ljmp Mod2               ; und los geht's

att_D:  jb E2in,att_H           ; E3 nicht in | springe wenn E2 nicht in
        jnb E1in,att_G          ; springe wenn E1 in (E1 & E2 in)
        clr Ansch               ; E3 nicht in, E2 in, E1 nicht in
        jnb MMon,att_D1         ; springe wenn kein Mousemode
        jb ScOn,att_D1          ; springe wenn kein Scanmode
        mov Modus,#M_7          ; Modus 7 setzen
        mov atten,#A_OK         ; Mitteilung löschen
        acall att_AA            ; LEDs in Ordnung bringen
        clr abusy               ; Attention verlassen
        ljmp Mod7               ; und los geht's

att_D1: mov Modus,#M_3          ; Modus 3 setzen
        mov atten,#A_OK         ; Mitteilung löschen
        acall att_AA            ; LEDs in Ordnung bringen
        clr abusy               ; Attention verlassen
        ljmp Mod3               ; und los geht's

att_H:  jb E1in,att_I           ; E3 nicht in, E2 nicht in | springe wenn E1 nicht in
        clr Ansch               ; E3 nicht in, E2 nicht in, E1 in
        jnb MMon,att_H1         ; springe wenn kein Mousemode
        jb ScOn,att_H1          ; springe wenn kein Scanmode
        mov Modus,#M_6          ; Modus 6 setzen
        mov atten,#A_OK         ; Mitteilung löschen
        acall att_AA            ; LEDs in Ordnung bringen
        clr abusy               ; Attention verlassen
        ljmp Mod6               ; und los geht's

att_H1: mov Modus,#M_1          ; Modus 1 setzen
        mov atten,#A_OK         ; Mitteilung löschen
```

```
        acall att_AA            ; LEDs in Ordnung bringen
        clr abusy               ; Attention verlassen
        ljmp Mod1               ; und los geht's

att_I:  clr Ansch               ; nichts in
        jnb MMon,att_I1         ; springe wenn kein Mousemode
        jb ScOn,att_I1          ; springe wenn kein Scanmode
        mov Modus,#M_8          ; Modus 8 setzen
        mov atten,#A_OK         ; Mitteilung löschen
        acall att_AA            ; LEDs in Ordnung bringen
        clr abusy               ; Attention verlassen
        clr TelOn               ; kein Telefonmodus
        ljmp Mod8               ; und los geht's

att_I1: mov Modus,#M_5          ; Modus 5 setzen
        mov atten,#A_OK         ; Mitteilung löschen
        acall att_AA            ; LEDs in Ordnung bringen
        clr abusy               ; Attention verlassen
        clr TelOn               ; kein Telefonmodus
        ljmp Mod5               ; und los geht's

att_C:  cjne Modr,#M_1,att_C1   ; springe wenn Modus != M_1
        mov atten,#A_OK         ; Mitteilung löschen
        acall att_AA            ; LEDs in Ordnung bringen
        clr abusy               ; Attention verlassen
        ljmp Mod1               ; und los geht's

att_C1: cjne Modr,#M_2,att_C2   ; springe wenn Modus != M_2
        mov atten,#A_OK         ; Mitteilung löschen
        acall att_AA            ; LEDs in Ordnung bringen
        clr abusy               ; Attention verlassen
        ljmp Mod2               ; und los geht's

att_C2: cjne Modr,#M_3,att_C3   ; springe wenn Modus != M_3
        mov atten,#A_OK         ; Mitteilung löschen
        acall att_AA            ; LEDs in Ordnung bringen
        clr abusy               ; Attention verlassen
        ljmp Mod3               ; und los geht's

att_C3: cjne Modr,#M_4,att_C4   ; springe wenn Modus != M_4
        mov atten,#A_OK         ; Mitteilung löschen
        acall att_AA            ; LEDs in Ordnung bringen
        clr abusy               ; Attention verlassen
        ljmp Mod4               ; und los geht's

att_C4: cjne Modr,#M_5,att_C5   ; springe wenn Modus != M_5
        mov atten,#A_OK         ; Mitteilung löschen
        acall att_AA            ; LEDs in Ordnung bringen
        clr abusy               ; Attention verlassen
        clr TelOn               ; kein Telefonmodus
        ljmp Mod5               ; und los geht's

att_C5: cjne Modr,#M_6,att_C6   ; springe wenn Modus != M_6
        mov atten,#A_OK         ; Mitteilung löschen
        acall att_AA            ; LEDs in Ordnung bringen
        clr abusy               ; Attention verlassen
        ljmp Mod6               ; und los geht's

att_C6: cjne Modr,#M_7,att_C7   ; springe wenn Modus != M_7
        mov atten,#A_OK         ; Mitteilung löschen
        acall att_AA            ; LEDs in Ordnung bringen
        clr abusy               ; Attention verlassen
        ljmp Mod7

att_C7: cjne Modr,#M_8,att_C8   ; springe wenn Modus != M_8
        mov atten,#A_OK         ; Mitteilung löschen
        acall att_AA            ; LEDs in Ordnung bringen
        clr abusy               ; Attention verlassen
        clr TelOn               ; kein Telefonmodus
        ljmp Mod8

att_C8: setb Fat3               ; fatal error 3 setzen
        clr abusy               ; Attention verlassen
        ljmp FatalError         ; und das war's für immer

att_B:  cjne att,#A_Mod,att_J   ; springe wenn atten != A_Mod
        mov Modus,#M_New        ; neuer Modus muß gesucht werden
        ajmp att_A              ; such ihn

att_J:  cjne att,#A_Prog,att_K  ; springe wenn atten != A_Prog
        clr abusy               ; Attention verlassen
```

```
        ljmp Prog               ; Programmierung ermöglichen

att_K:  cjne att,#A_HW,att_L    ; springe wenn atten != A_HW
        mov Modus,#M_New        ; neuer Modus muß gesucht werden
        mov Port5,P5            ; Port5 aktualisieren
        jnb HM,att_K1           ; springe wenn HW-Mouse off
        setb MMon               ; Mousemodus on
        ajmp att_A              ; zum Modus zurück

att_K1: clr MMon                ; Mousemodus off
        anl Latch1,#10001111b   ; alle Mousetasten aus
        lcall mL1               ; Latch 1 aktualisieren
        ajmp att_A              ; zum Modus zurück

att_L:  cjne att,#A_SWSM,att_M  ; springe wenn atten != A_SWSM
        mov Modus,#M_New        ; neuer Modus muß gesucht werden
        setb MMon               ; Mousemode on
        ajmp att_A              ; zum Modus zurück

att_M:  cjne att,#A_SWMS,att_M1 ; springe wenn atten != A_SWMS
        mov Modus,#M_New        ; neuer Modus muß gesucht werden
        anl Latch1,#10001111b   ; alle Mousetasten aus
        lcall mL1               ; Latch 1 aktualisieren
        clr MMon                ; Mousemode off
        ajmp att_A              ; zum Modus zurück

att_M1: cjne att,#A_Warm,att_M2 ; springe wenn atten != A_Warm
        acall att_M3            ; auf Reset vorbereiten
        ljmp IWarm              ; Warm-Reset

att_M2: cjne att,#A_Cold,att_N  ; springe wenn atten != A_Cold
        acall att_M3            ; auf Reset vorbereiten
        ljmp ICold              ; Cold-Reset

att_M3: mov atten,#M_New        ; alles wiederherstellen
        clr abusy
        clr EA                  ; alle Interrupts verbieten
        mov PSW,#0              ; PSW auf Anfang
        ret

att_N:  setb Fat4               ; Fatal Error 4 setzen
        clr abusy               ; Attention verlassen
        ljmp FatalError         ; und weg ist er

; lokale Unterroutine zum Initialieren der LEDs

att_AA: jb MMon,att_AB          ; springe wenn Mousemode
        mov Latch0,#0FFh        ; alle Mouse-LEDs off
        lcall mL0               ; Latch 0 aktualisieren
        orl Latch1,#00001111b   ; alle Ausgangsselektor-LEDs off
        cjne soutr,#G1,att_AC   ; springe wenn sout != G1
        clr A1Sel               ; A1Sel-LED on
        sjmp att_AG             ; zum nächsten Sammelpunkt

att_AC: cjne soutr,#G2,att_AD   ; springe wenn sout != G2
        clr A2Sel               ; A2Sel-LED on
        sjmp att_AG             ; zum nächsten Sammelpunkt

att_AD: cjne soutr,#G3,att_AE   ; springe wenn sout != G3
        clr A3Sel               ; A3Sel-LED on
        sjmp att_AG             ; zum nächsten Sammelpunkt

att_AE: cjne soutr,#M_On,att_AF ; springe wenn sout != M_On
        clr MOn                 ; MOn-LED on
        sjmp att_AG             ; zum nächsten Sammelpunkt

att_AF: setb Fat1               ; FatalError 1 setzen
        ljmp FatalError         ; they never come back

att_AG: lcall mL1               ; Latch 1 aktualisieren
        ret                     ; das war’s

att_AB: orl Latch1,#00001111b   ; alle Ausgangsselektor-LEDs off
        lcall mL1               ; Latch 1 aktualisieren
        mov Latch0,#0FFh        ; alle LEDs off
        cjne moutr,#links,att_AH; springe wenn mout != links
        clr MLSel               ; MLSel LED on
        sjmp att_AL             ; zum nächsten Sammelpunkt

att_AH: cjne moutr,#mitte,att_AI; springe wenn mout != mitte
        clr MMSel               ; MMSel LED on
```

```
        sjmp att_AL             ; zum nächsten Sammelpunkt

att_AI: cjne moutr,#rechts,att_AJ;springe wenn mout != rechts
        clr MRSel               ; MRSel LED on
        sjmp att_AL             ; zum nächsten Sammelpunkt

att_AJ: cjne moutr,#M_Off,att_AK; springe wenn mout != M_Off
        clr MOff                ; MOff LED on
        sjmp att_AL             ; zum nächsten Sammelpunkt

att_AK: setb Fat5               ; FatalError 5 setzen
        ljmp FatalError         ; und tot ist er

att_AL: cjne mselr,#click,att_AM; springe wenn msel != click
        clr MC                  ; MC LED on
        sjmp att_AQ             ; zum nächsten Sammelpunkt

att_AM: cjne mselr,#dclick,att_AN;springe wenn msel != dclick
        clr MDC                 ; MDC LED on
        sjmp att_AQ             ; zum nächsten Sammelpunkt

att_AN: cjne mselr,#switch,att_AO;springe wenn msel != switch
        clr MS                  ; MS LED on
        sjmp att_AQ             ; zum nächsten Sammelpunkt

att_AO: cjne mselr,#nextm,att_AP;springe wenn msel != nextm
        clr Next                ; Next LED on
        sjmp att_AQ             ; zum nächsten Sammelpunkt

att_AP: setb Fat2               ; FatalError 2 setzen
        ljmp FatalError         ; da geht er hin...

att_AQ: lcall mL0               ; Latch 0 aktualisieren
        ret                     ; das war's
```

Der erste Teil dieser Routine (ab `att_A`) sucht aus den derzeit eingesteckten Sensoren, den DIP–Schaltern und Kennbits den neuen Modus aus und verzweigt in das entsprechende Modul. Während des „Aufenthalts“ in dieser Routine ist die Messageauslösung durch den Timer–Interrupt gesperrt, um nicht durch eine Kollision zweier Messages Fehler zu erzeugen.

Der zweite Teil der Routine (ab `att_B`) wertet die Message aus und springt — wenn nötig — zu `att_A` um den Modus neu zu bestimmen, oder löst einen Reset aus.

Da nach einem Wechsel vom Switchmode in den Mousemode (oder umgekehrt) die LEDs des „alten“ Modes ausgeschaltet werden, müssen die LED–Einstellungen des neuen Moduses nach den Kennwerten des Moduses (z.B. `sout, mout, msel, ...`) neu berechnet werden; dies geschieht ab `att_AA`.

9.7.5 Programmieren

```
 ; Programmieren
 ; *************

Prog:   lcall StopInt           ; fast alle Interrupts unterbinden
        setb PrOn               ; Programmode einschalten
        mov C,A1                ; A1 sichern
        mov oA1,C
        clr A1                  ; A1 löschen
        mov BL0,Latch0          ; Latch 0 sichern
        mov Latch0,#11111110b   ; nur eine LED on (zum Rotieren)
        lcall mL0               ; Latch 0 aktualisieren
        mov BL1,Latch1          ; Latch 1 sichern
        mov Latch1,#10001111b   ; alle LEDs off
        lcall mL1               ; Latch 1 aktualisieren
        mov ODIP,DIP            ; DIP-Stellungen sichern
```

```
        acall Pr_Kar            ; Karrenzzeit neu einstellen
Pro_A:  mov A,ODIP              ; alte DIP-Einstellungen für den Vergleich in A
        cjne A,DIP,Pro_BB       ; springen wenn DIP verändert
        mov Port5,P5            ; Port 5 aktualisieren
        jb HMin,Pro_D           ; springe wenn HW nicht in
        jnb E1in,Pro_E          ; springe wenn E1 in
        jnb E2in,Pro_E          ; springe wenn E2 in
        jnb E3in,Pro_E          ; springe wenn E3 in
        clr Ansch               ; Anschlußfehler löschen
        clr ET0                 ; Timer0-Interrupt verbieten
        orl ADCON,#Bit0         ; ADC-Eingang 7 wählen
        orl ADCON,#Bit3         ; ADC starten
Pro_G:  mov A,ADCON             ; ADCON in A zur Bitabfrage
        jnb Acc.4,Pro_G         ; Warten auf ADC-Wandlung
        anl ADCON,#nBit4        ; Interruptflag löschen
        setb ET0                ; Timer0-Interrupt erlauben
        mov stime,ADCH          ; Ergebnis speichern
        sjmp Pro_A              ; und von vorne

Pro_BB: ajmp Pro_B              ; Sprungverlängerung

Pro_E:  setb Ansch              ; Anschlußfehler setzen
        sjmp Pro_A              ; auf Besserung warten

Pro_D:  jb E1in,Pro_F           ; HM nicht in | springe wenn E1 nicht in
        jnb E2in,Pro_E          ; springe wenn E2 in
        jnb E3in,Pro_E          ; springe wenn E3 in
        clr Ansch               ; Anschlußfehler löschen
        jnb E1,Pro_A            ; springe wenn E1 nicht gedrückt
        lcall Entprell          ; Entprellen
        clr A1Sel               ; A1Sel LED on
        lcall mL1               ; Latch1 aktualisieren
        mov ticks,#0            ; ticks löschen
        clr tcar                ; ticks carry löschen
Pro_H:  jnb E1,Pro_I            ; springen wenn E1 nicht mehr gedrückt
        jnb tcar,Pro_H          ; springen wenn noch kein Überlauf
        clr ET0                 ; Timer-Interrupt verbieten
        lcall LEDsOn            ; alle LEDs on
Pro_J:  jb E1,Pro_J             ; Warten auf wieder Loslassen
        lcall LEDsOff           ; alle LEDs wieder normal
        setb ET0                ; Timer-Interrupt erlauben
        sjmp Pro_I0             ; keine Änderung, alles von vorne

Pro_I:  mov atime,ticks         ; Zeit speichern ##Weitere Sicherheitsabfragen möglich
Pro_I0: setb A1Sel              ; A1Sel-LED off
        lcall mL1               ; Latch1 aktualisieren
        sjmp Pro_A              ; alles von vorne

Pro_F:  jb E2in,Pro_E           ; HM und E1 nicht in | springe wenn E2 nicht in
        jnb E3in,Pro_E          ; springe wenn E3 in
        clr Ansch               ; Anschlußfehler löschen
        jnb E2,Pro_A            ; springe wenn E2 nicht gedrückt
        lcall Entprell          ; Entprellen
        setb A1                 ; Ausgang zum Test ein
        mov ticks,#0            ; ticks löschen
        clr tcar                ; ticks carry löschen
Pro_K:  jnb E2,Pro_L            ; springen wenn E2 nicht mehr gedrückt
        jnb tcar,Pro_K          ; springe wenn noch kein Überlauf
        clr ET0                 ; Timer-Interrupt verbieten
        lcall LEDsOn            ; alle LEDs on
Pro_M:  jb E2,Pro_M             ; warten auf wieder Loslassen
        lcall LEDsOff           ; alle LEDs wieder normal
        setb ET0                ; Timer-Interrupt erlauben
        sjmp Pro_L0             ; keine Änderung, alles wieder von vorne

Pro_L:  mov otime,ticks         ; Zeit speichern ##Weitere Sicherheitsabfragen möglich
Pro_L0: clr A1                  ; A1 wieder löschen
        ajmp Pro_A              ; alles von vorne

Pro_B:  jb PrgOn,Pr_End         ; Ende der Programmierung
        anl A,#KBit             ; Karr-Bits herausschneiden
        mov R0,A                ; alten Wert ins R0
        mov A,DIP               ; neue Stellung in A
        mov ODIP,DIP            ; neue Stellung speichern
        anl A,#KBit             ; Karr-Bits herausschneiden
        cjne A,R0_0,Pro_C       ; springe wenn neue Stellung != alte Stellung
        ajmp Pro_A              ; falscher Alarm (unwichtige Änderung)

Pro_C:  acall Pr_Kar            ; neue Karrenzzeit einstellen
        ajmp Pro_A              ; und von vorne
```

```
Pr_End: clr Ansch               ; zur Sicherheit Anschlußfehler löschen
        mov C,oA1               ; A1 restoren
        mov A1,C
        mov Latch0,BL0          ; Latch 0 restoren
        lcall mL0               ; Latch 0 aktualisieren
        mov Latch1,BL1          ; Latch 1 restoren
        lcall mL1               ; Latch 1 aktualisieren
        mov Modus,#M_New        ; neuer Modus!
        mov atten,#A_Mod        ; neuer Modus!
        clr PrOn                ; ProgrammModus aus
        clr RI                  ; kein serieller Eingang!
        clr TI                  ; kein serieller Ausgang!
        lcall StartInt          ; Interrupts wieder starten
        ljmp att_A              ; und zurück zum normalen Arbeiten

Pr_Kar: push Acc                ; A retten
        mov A,DIP               ; neuen DIP holen
        anl A,#KBit             ; Karr-Bits herausschneiden
        jz Pr_K1                ; springe wenn 1:1
        mov R0,A                ; Zahl ins Register 0
        mov A,atime             ; atime zur Bearbeitung in A
Pr_K2:  rr A                    ; durch 2 dividieren
        clr Acc.7               ; oberstes Bit löschen (Rotate!)
        djnz R0,Pr_K2           ; wiederholen bis R0 == 0
        mov ktime,A             ; Ergebnis sichern
        pop Acc                 ; A zurück
        ret                     ; das war's

Pr_K1:  mov ktime,atime         ; ktime = atime
        pop Acc                 ; A zurück
        ret                     ; das war's

END
```

Diese Routine ist zwar prinzipiell ein Modul und sollte daher im Kapitel 9.7.3 zu finden sein, hat aber eine Sonderstellung, da fast alle Interrupts verboten werden und die LED–Steuerung selbst übernommen wird.

Jede Einstellmöglichkeit wird in einer eigenen Unterroutine nach den Definitionen aus dem Kapitel 4.5 behandelt; sollte kein Sensor eingesteckt sein, oder mehr als einer, sodaß kein eindeutiger Programmiermodus zu erkennen ist, wird eine Fehlermeldung über die LEDs ausgegeben.

Kapitel 10

Entwicklung der Software für den PC

Im folgenden wird die Entwicklung des „einfachen“ Programmes „DoSwitch“auf einem PC beschrieben, das mir bei der Fehlersuche sehr behilflich war und gleichzeitig fast alle Funktionen der seriellen Schnittstelle des Multiswitches getestet hat. Dieses Programm entspricht nicht dem von mir im Kapitel 8.3.3.2 vorgeschlagenen Drei–Ebenen–System, da es sich nicht um eine Version für Servicetechniker handelt, sondern *ausschließlich* für mich als Entwickler als Unterstützung gedacht war.

10.1 Allgemeines

Da, wie später noch erläutert wird, die ganzen Tests der Soft– und Hardware auf einem *AMIGA* durchgeführt wurden, ist auch dieses Programm für einen *AMIGA* entwickelt worden. Es ist in Standard–C geschrieben und auf jedem *AMIGA* ab der ROM–Version 3.0 lauffähig, mit wenigen Änderungen der Graphik auch unter 2.04. Da jedoch sowohl die Unterstützung der seriellen Schnittstelle als auch der graphischen Oberfläche bei jedem Computer sehr betriebssystemnah ist, ist das Programm nur sehr bedingt auf andere Computer–Systeme portierbar.

Um eine einfache Bedienung bei den Tests zu ermöglichen, habe ich eine kleine graphische Oberfläche implementiert, sodaß eine kurze Beschreibung der Funktionen und Bedienung reicht, wie sie im Kapitel 10.3 zu finden ist.

10.2 das Programm „DoSwitch“

In diesem Abschnitt wird das Programm in groben Zügen beschrieben. Vorkenntnisse in C und dem Umgang mit dem Betriebssystem OS 3.0 werden im folgenden vorausgesetzt, da

die detaillierte Beschreibung für jemanden ohne Vorkenntnisse den Rahmen dieser Diplomarbeit sprengen würde. Zum besseren Verständnis des Programmes sei auf die Screenshots im Kapitel 10.3 hingewiesen.

10.2.1 Definitionen

10.2.1.1 Includes

```
/*

  DoSwitch

  Erstes, einfaches Programm zum Steuern des Multiswitches

  History:

  1.0: Erstausgabe

*/

#include <intuition/intuition.h>
#include <intuition/gadgetclass.h>

#include <libraries/gadtools.h>
#include <libraries/dos.h>
#include <libraries/dosextens.h>

#include <exec/types.h>
#include <devices/serial.h>
#include <dos/dos.h>
#include <stdlib.h>
#include <string.h>

#include <proto/dos.h>
#include <proto/exec.h>
#include <proto/intuition.h>
#include <proto/gadtools.h>
```

Nach dem für C üblichen Kopf folgen die Includes für die Systemroutinen; Intuition ist für alle Window– und Screenfunktionen zuständig, Gadtools für alle Gadgets, Exec für alle Betriebssystemfunktionen, Serials für die serielle Schnittstelle und die Protos werden benötigt, da ich den Compiler auf Übergabe der Parameter in den Registern eingestellt habe.

10.2.1.2 globale Variablendefinitionen

Zunächst der für jedes *AMIGA*–Programm übliche Versionsstring:

```
/*------------------ Version ------------------------------------------*/

char VERSION[]="\0$VER: DoSwitch V1.0 by CHB on 19.04.97";

/*---------------------------------------------------------------------*/
```

Danach zu den Structs:

```
/* Structs */

struct Screen *myscreen = NULL;
struct Window *mywindow = NULL;
struct Gadget *mygadgetbase = NULL;
```

```
struct Gadget *myList = NULL;
struct Gadget *mySlider = NULL;
struct Gadget *myWrite = NULL;
struct Gadget *myCycle = NULL;
struct Gadget *myRead = NULL;
struct Gadget *myReset = NULL;
struct Gadget *myInfo = NULL;
struct Gadget *myConnect = NULL;
struct Gadget *mygadget = NULL;
struct Gadget *myNum1 = NULL;
struct Gadget *myNum2 = NULL;
struct Gadget *myNum3 = NULL;
struct NewGadget mynewgadget;
struct IntuiMessage *myMsg = NULL;

struct MsgPort *SerialMP = NULL; /* Port für io messages */
struct IOExtSer *SerialIO = NULL;   /* Datastructure für das device */
struct IOExtSer *IncomingIO = NULL; /* Structure für einlaufende Zeichen */

struct EasyStruct myeasy = {sizeof(struct EasyStruct), 0,NULL,"","OK"};
```

Die Variable `myscreen` benötige ich, um später aktuelle Informationen über den PublicScreen zu erhalten, `mywindow` ist der Basiszeiger auf mein Window, alle folgenden Variablen vom Typ `struct Gadget` sind die Zeiger auf Gadgetstrukturen (für jedes Gadget eine Struktur), `mynewgadget` benötige ich als Basisstruktur für die Gadgets und `myMsg` ist ein Messageport für die Kommunikation mit Intuition für die Gadgets.
Danach drei Strukturen für die Kommunikation mit dem seriellen Device, zwei Devicestrukturen (`SerialIO, IncomingIO`) und wieder ein Messageport, aber dieses mal für die Kommunikation mit dem Device.
Zuletzt folgt noch eine Struktur (`myeasy`) für den Aufbau von Requestern bei Fehlermeldungen.

Anschließend die restlichen globalen Variablen:

```
/* restliche Globale */

APTR vi;
int SerOpen = 0;        /* zur Kennzeichnung eines offenen Devices */
long myReg=0;
unsigned char Buffer[3];
```

`vi` wird für die Informationen des Screens benötigt (VisualInfo), `SerOpen` ist eine Boolvariable zur Erkennung des Zustandes des seriellen Devices (mangels einer Basisadresse), `myReg` wird zur Zahlenabfrage von Gadgets verwendet und `Buffer` dient als Ein/Ausgabebuffer für die serielle Schnittstelle.

Zu guter letzt noch ein Struct für den zu verwendenden Text:

```
struct TextAttr topaz80 =
{
    "topaz.font",
    8,
    0,
    0,
};
```

10.2.2 Funktionen

In diesem Kapitel werden alle Funktionen in der Reihenfolge des Programms aufgelistet und erläutert.

10.2.2.1 machszu

```
/* sicheres Schließen aller Resourcen */

void machszu(int code)
{
   if (SerOpen) CloseDevice((struct IORequest *) SerialIO);
   if (IncomingIO) DeleteExtIO((struct IORequest *) IncomingIO);
   if (SerialIO) DeleteExtIO((struct IORequest *) SerialIO);
   if (SerialMP) DeletePort(SerialMP);
   if (mywindow) CloseWindow(mywindow);
   if (mygadgetbase) FreeGadgets(mygadgetbase);
   if (vi) FreeVisualInfo(vi);
   if (myscreen) UnlockPubScreen(NULL,myscreen);
   if (GadToolsBase) CloseLibrary((struct Library *) GadToolsBase);
   if (IntuitionBase) CloseLibrary((struct Library *) IntuitionBase);
   exit(code);
}
```

Diese Funktion übernimmt als Argument die an das Betriebssystem zu übergebende Fehlernummer und schließt alle Resourcen in jedem beliebigen Zustand des Programmes, sodaß auch bei jedem schweren Fehler damit ausgestiegen werden kann.

10.2.2.2 ShowNum

```
/* Returnwerte anzeigen */

void ShowNum(int Anz)
{
   switch(Anz)
   {
      case 1:
         GT_SetGadgetAttrs(myNum1,mywindow,NULL,
         GTNM_Format,"%ld",
         GTNM_Number,Buffer[0],
         TAG_DONE);
         GT_SetGadgetAttrs(myNum2,mywindow,NULL,
         GTNM_Format,"",
         TAG_DONE);
         GT_SetGadgetAttrs(myNum3,mywindow,NULL,
         GTNM_Format,"",
         TAG_DONE);
         break;
      case 2:
         GT_SetGadgetAttrs(myNum1,mywindow,NULL,
         GTNM_Format,"%ld",
         GTNM_Number,Buffer[0],
         TAG_DONE);
         GT_SetGadgetAttrs(myNum2,mywindow,NULL,
         GTNM_Format,"%ld",
         GTNM_Number,Buffer[1],
         TAG_DONE);
         GT_SetGadgetAttrs(myNum3,mywindow,NULL,
         GTNM_Format,"",
         TAG_DONE);
         break;
      case 3:
         GT_SetGadgetAttrs(myNum1,mywindow,NULL,
         GTNM_Format,"%ld",
         GTNM_Number,Buffer[0],
         TAG_DONE);
         GT_SetGadgetAttrs(myNum2,mywindow,NULL,
         GTNM_Format,"%ld",
         GTNM_Number,Buffer[1],
         TAG_DONE);
         GT_SetGadgetAttrs(myNum3,mywindow,NULL,
         GTNM_Format,"%ld",
         GTNM_Number,Buffer[1],
         TAG_DONE);
         break;
      default:
         GT_SetGadgetAttrs(myNum1,mywindow,NULL,
         GTNM_Format,"",
         TAG_DONE);
```

```
         GT_SetGadgetAttrs(myNum2,mywindow,NULL,
         GTNM_Format,"",
         TAG_DONE);
         GT_SetGadgetAttrs(myNum3,mywindow,NULL,
         GTNM_Format,"",
         TAG_DONE);
   }
}
```

Wie später noch gezeigt wird, habe ich drei Nummerngadgets definiert, die dazu dienen, die vom seriellen Device empfangenen Daten direkt ohne Vorbearbeitung sehen zu können.

Als Argument übergibt man der Funktion die Anzahl der gültigen Bytes im Buffer. Die Funktion zeigt diese Daten an, die restlichen Gadgets werden geleert.

Da man eine Nummer nicht einfach direkt in ein Gadget schreiben kann, muß man die anzuzeigende Zahl als neues Attribut des Gadgets definieren; dazu dient die Funktion `GT_SetGadgetAttrs`. Um die nicht benötigten Gadgets auf „leer“ zu stellen, werden die Anzeigeparameter auf `""` gesetzt, da es keine direkte Funktion dafür gibt.

10.2.2.3 SWrite

```
/* serielles Senden */

void SWrite(int len)
{
   SerialIO->IOSer.io_Command = CMD_WRITE;
   SerialIO->IOSer.io_Length = len;
   SerialIO->IOSer.io_Data = (APTR) Buffer;
   if (DoIO((struct IORequest *) SerialIO))
   {
      myeasy.es_TextFormat="---======* ERROR *********\nKann die Daten nicht senden!";
      EasyRequest(mywindow,&myeasy,NULL);
      machszu(10);
   }
}
```

Diese Funktion übernimmt als Argument die Anzahl der im Buffer enthaltenen Datenbytes, die Übertragen werden sollen, sendet diese Daten und bricht das Programm ab, wenn das Device eine Fehlermeldung zurückschickt.

Da das Device vom Prinzip her ein Objekt ist, kann darauf nur mit im Device definierten Methoden zugegriffen werden. Um eine Methode zu starten, muß man eine IOStruktur (`SerialIO`) ausfüllen und an das Device senden (mit `DoIO`). Sollte eine Fehlermeldung beim Senden entstehen, wird ein Requester mit der entsprechenden Fehlermeldung aufgebaut (`EasyRequest`) und danach das Programm beendet.

10.2.2.4 SRead

```
/* serielles Empfangen */

void SRead(int len)
{
   IncomingIO->IOSer.io_Command = CMD_READ;
   IncomingIO->IOSer.io_Length = len;
   IncomingIO->IOSer.io_Data = (APTR) Buffer;
   if (DoIO((struct IORequest *) IncomingIO))
   {
      myeasy.es_TextFormat="---======* ERROR *********\nKann die Daten nicht empfangen!";
```

```
      EasyRequest(mywindow,&myeasy,NULL);
      machszu(10);
   }
   ShowNum(len);
}
```

Diese Funktion arbeitet genauso wie `SWrite`, bis auf die Unterschiede, daß bei positiver Erledigung die Daten gelesen und automatisch in den Nummerngadgets angezeigt werden.

10.2.2.5 GadOn

```
/* Gadgets on */

void GadOn(void)
{
   GT_SetGadgetAttrs(myWrite,mywindow,NULL,
   GA_Disabled,FALSE,
   TAG_DONE);
   GT_SetGadgetAttrs(myList,mywindow,NULL,
   GA_Disabled,FALSE,
   TAG_DONE);
   GT_SetGadgetAttrs(mySlider,mywindow,NULL,
   GA_Disabled,FALSE,
   TAG_DONE);
   GT_SetGadgetAttrs(myReset,mywindow,NULL,
   GA_Disabled,FALSE,
   TAG_DONE);
   GT_SetGadgetAttrs(myCycle,mywindow,NULL,
   GA_Disabled,FALSE,
   TAG_DONE);
   GT_SetGadgetAttrs(myRead,mywindow,NULL,
   GA_Disabled,FALSE,
   TAG_DONE);
}
```

Diese Funktion schaltet alle Gadgets außer „Connect“ ein.

10.2.2.6 GadOff

```
/* Gadgets off */

void GadOff(void)
{
   GT_SetGadgetAttrs(myWrite,mywindow,NULL,
   GA_Disabled,TRUE,
   TAG_DONE);
   GT_SetGadgetAttrs(myList,mywindow,NULL,
   GA_Disabled,TRUE,
   TAG_DONE);
   GT_SetGadgetAttrs(mySlider,mywindow,NULL,
   GA_Disabled,TRUE,
   TAG_DONE);
   GT_SetGadgetAttrs(myReset,mywindow,NULL,
   GA_Disabled,TRUE,
   TAG_DONE);
   GT_SetGadgetAttrs(myCycle,mywindow,NULL,
   GA_Disabled,TRUE,
   TAG_DONE);
   GT_SetGadgetAttrs(myRead,mywindow,NULL,
   GA_Disabled,TRUE,
   TAG_DONE);
}
```

Diese Funktion schaltet alle Gadgets außer „Connect“ aus.

10.2.2.7 ReadReg

```
/* Prozedure zum Lesen des eingestellten Registers */
/* test: wenn true, werden Fehlermeldungen durchgeführt */

void ReadReg(int test)
{
      Buffer[0]=192; /* Read-Befehl */
      SWrite(1);
      SRead(1);
      if (Buffer[0])
      {
         if (test)
         {
            GT_SetGadgetAttrs(myInfo,mywindow,NULL,
            GTTX_Text,"Lesefehler!",
            TAG_DONE);
         }
         return;
      }
      else /* alles OK */
      {  /* Holen des zu lesenden Registers */
         if(!(GT_GetGadgetAttrs(myList,mywindow,NULL,
         GTMX_Active,&myReg,
         TAG_DONE))) machszu(20);
         Buffer[0]=myReg+1;
         SWrite(1);  /* Register senden */
         SRead(1);
         GT_SetGadgetAttrs(mySlider,mywindow,NULL,
         GTSL_Level,Buffer[0],
         TAG_DONE);
         if (test)
         {
            GT_SetGadgetAttrs(myInfo,mywindow,NULL,
            GTTX_Text,"Lesen OK!",
            TAG_DONE);
         }
         return;
      }
}
```

Mit dieser Funktion wird das aktuell mit dem MutualExklude–Gadget eingestellte Register vom Multiswitch gelesen. Als Argument ist eine Boolvariable zu übergeben: wenn diese Variable `TRUE` ist, werden etwaige Fehlermeldungen im Info–Gadget ausgegeben, sonst nicht.

Realisiert wird diese Funktion wie folgt: Zuerst wird der Schreibbefehl an den Multiswitch gesendet. Wird dieser positiv beantwortet, so wird mittels `GT_GetGadgetAttrs` das aktuelle Register aus dem MutualExklude–Gadget ausgelesen und an den Multiswitch gesendet; danach wird das Ergebnis im Slider–Gadget angezeigt. Beim Auftreten irgendwelcher Fehler vom Multiswitch wird eine entsprechende Fehlermeldung im Info–Gadget ausgegeben, ansonsten wird eine positive Meldung angezeigt.

10.2.2.8 DoConnect

```
/* Verbindung aufbauen */

void DoConnect(void)
{
   Buffer[0]=0xff;      /* Resetzeichen */
   Buffer[1]=2;      /* PC-Kennung */
   SWrite(2);     /* Senden */
   SRead(2);      /* Antwort empfangen */

/* Antwort testen */

   if ((Buffer[0]==0)&&(Buffer[1]==0))
```

```
   {
      SWrite(1);  /* positive Antwort */

/* Connect disablen und Info ausgeben */

      GadOn();
      GT_SetGadgetAttrs(myConnect,mywindow,NULL,
      GA_Disabled,TRUE,
      TAG_DONE);
      GT_SetGadgetAttrs(myInfo,mywindow,NULL,
      GTTX_Text,"Verbindung OK",
      TAG_DONE);
      ReadReg(0);
   }
   else
   {

/* Write, List disablen und Info ausgeben */

      GadOff();
      GT_SetGadgetAttrs(myConnect,mywindow,NULL,
      GA_Disabled,FALSE,
      TAG_DONE);
      GT_SetGadgetAttrs(myInfo,mywindow,NULL,
      GTTX_Text,"keine Verbindung!",
      TAG_DONE);
   }
}
```

Mit dieser Funktion wird die Verbindung zum Multiswitch aufgebaut. Zunächst sendet die Funktion das Resetzeichen und die Kennung für einen PC an den Multiswitch. Sodann wird auf die Antwort des Multiswitches gewartet und diese ausgewertet. Sollte die Antwort dem Protokoll entsprechen, wird eine positive Bestätigung an den Multiswitch gesendet, alle Gadgets aktiviert und das Connect–Gadget deaktiviert. Zuletzt wird noch der aktuelle Wert des angezeigten Registers gelesen, angezeigt und eine OK–Meldung ausgegeben. Bei einem negativen Ergebnis werden die Gadgets deaktiviert und das Connect–Gadget aktiviert, um einen erneuten Versuch des Verbindungsaufbaus starten zu können; weiters wird eine Fehlermeldung im Info–Gadget angezeigt.

10.2.3 Hauptteil

10.2.3.1 Kopf und Variable

```
/* Hauptstück */

void main(void)
{

/* Variable */

   long myVal=0;
   struct TextAttr *mytext = &topaz80;
   STRPTR TheLabels[]=
   {"P1","P3","P4","P5","Latch0","Latch1",
    "Modus","mout","msel","sout","atten","Error",
    "LError","Misc","Prell","ctime","dtime","otime",
    "atime","ktime","stime","ABatt",NULL};
   STRPTR TheCycle[]=
   {"Warm","Cold",NULL};
```

Die Variable `myVal` wird zum Auslesen des Slider–Gadgets benötigt, die Struktur `mytext` enthält die Textattribute für das Window. Danach folgen die Definitionen der Labels für das MutualExklude–Gadget und das Cycle–Gadget.

10.2.3.2 Librarys öffnen

```
   IntuitionBase = (struct IntuitionBase *) OpenLibrary("intuition.library", 39L);
   if (!(IntuitionBase)) machszu(20);

   GadToolsBase = (struct Library *) OpenLibrary("gadtools.library", 36L);
   if (!(GadToolsBase)) machszu(20);
```

Die benötigten Librarys werden geöffnet. Wenn dabei ein Fehler auftritt, wird mit einem Fehlercode für schwere Fehler abgebrochen.

10.2.3.3 Gadgets erstellen

```
/* Visualinfo holen */

   myscreen = LockPubScreen(NULL);
   if (!(vi = GetVisualInfo(myscreen,NULL))) machszu(20);

/* Gadgetbasis herstellen */

   mygadgetbase=NULL;
   if (!(mygadgetbase=CreateContext(&mygadgetbase))) machszu(20);

/* Listengadget erstellen */

   mynewgadget.ng_LeftEdge=2;
   mynewgadget.ng_TopEdge=2;
   mynewgadget.ng_Width=70;
   mynewgadget.ng_Height=190;
   mynewgadget.ng_GadgetText="";
   mynewgadget.ng_TextAttr=mytext;
   mynewgadget.ng_GadgetID=1;
   mynewgadget.ng_Flags=PLACETEXT_RIGHT;
   mynewgadget.ng_VisualInfo=vi;
   mynewgadget.ng_UserData=NULL;

   if(!(myList=CreateGadget(MX_KIND,mygadgetbase,&mynewgadget,
   GA_Disabled,TRUE,
   GTMX_Labels,TheLabels,
   GTMX_Spacing,1,
   TAG_DONE))) machszu(20);

/* Slidergadget erstellen */

   mynewgadget.ng_LeftEdge=85;
   mynewgadget.ng_TopEdge=2;
   mynewgadget.ng_Width=25;
   mynewgadget.ng_Height=190;
   mynewgadget.ng_GadgetID=2;

   if(!(mySlider=CreateGadget(SLIDER_KIND,myList,&mynewgadget,
   GA_Disabled,TRUE,
   GTSL_Min,0,
   GTSL_Max,255,
   GTSL_MaxLevelLen,3,
   GTSL_Level,127,
   GTSL_LevelFormat, "%ld",
   GTSL_LevelPlace,PLACETEXT_BELOW,
   PGA_Freedom,LORIENT_VERT,
   TAG_DONE))) machszu(20);

/* Writegadget erstellen */

   mynewgadget.ng_LeftEdge=120;
   mynewgadget.ng_TopEdge=2;
   mynewgadget.ng_Width=180;
   mynewgadget.ng_Height=15;
   mynewgadget.ng_GadgetText="Write";
   mynewgadget.ng_GadgetID=3;
   mynewgadget.ng_Flags=PLACETEXT_IN;

   if(!(myWrite=CreateGadget(BUTTON_KIND,mySlider,&mynewgadget,
   GA_Disabled,TRUE,
   TAG_DONE))) machszu(20);
```

```
/* Readgadget erstellen */

   mynewgadget.ng_TopEdge=22;
   mynewgadget.ng_GadgetText="Read";
   mynewgadget.ng_GadgetID=8;
   mynewgadget.ng_Flags=PLACETEXT_IN;

   if(!(myRead=CreateGadget(BUTTON_KIND,myWrite,&mynewgadget,
   GA_Disabled,TRUE,
   TAG_DONE))) machszu(20);

/* Resetgadget erstellen */

   mynewgadget.ng_TopEdge=42;
   mynewgadget.ng_GadgetText="Reset";
   mynewgadget.ng_Width=88;
   mynewgadget.ng_GadgetID=9;
   mynewgadget.ng_Flags=PLACETEXT_IN;

   if(!(myReset=CreateGadget(BUTTON_KIND,myRead,&mynewgadget,
   GA_Disabled,TRUE,
   TAG_DONE))) machszu(20);


/* Reset Cycle Gadget erstellen */

   mynewgadget.ng_LeftEdge=212;
   mynewgadget.ng_TopEdge=42;
   mynewgadget.ng_GadgetText="";
   mynewgadget.ng_Width=88;
   mynewgadget.ng_GadgetID=10;

   if(!(myCycle=CreateGadget(CYCLE_KIND,myReset,&mynewgadget,
   GA_Disabled,TRUE,
   GTCY_Labels,TheCycle,
   TAG_DONE))) machszu(20);

/* Infogadget erstellen */

   mynewgadget.ng_LeftEdge=120;
   mynewgadget.ng_TopEdge=90;
   mynewgadget.ng_Width=180;
   mynewgadget.ng_GadgetText="Info";
   mynewgadget.ng_GadgetID=4;
   mynewgadget.ng_Flags=PLACETEXT_ABOVE;

   if(!(myInfo=CreateGadget(TEXT_KIND,myCycle,&mynewgadget,
   GTTX_Justification,GTJ_CENTER,
   GTTX_Text,"Initialisiere...",
   GTTX_Border,TRUE,
   TAG_DONE))) machszu(20);

/* Connectgadget erstellen */

   mynewgadget.ng_TopEdge=185;
   mynewgadget.ng_GadgetText="Connect";
   mynewgadget.ng_GadgetID=5;
   mynewgadget.ng_Flags=PLACETEXT_IN;

   if(!(myConnect=CreateGadget(BUTTON_KIND,myInfo,&mynewgadget,
   GA_Disabled,TRUE,
   TAG_DONE))) machszu(20);

/* Num1 Gadget erstellen */

   mynewgadget.ng_LeftEdge=120;
   mynewgadget.ng_TopEdge=110;
   mynewgadget.ng_Width=58;
   mynewgadget.ng_GadgetText="1";
   mynewgadget.ng_GadgetID=6;
   mynewgadget.ng_Flags=PLACETEXT_BELOW;

   if(!(myNum1=CreateGadget(NUMBER_KIND,myConnect,&mynewgadget,
   GTNM_Justification,GTJ_CENTER,
   GTNM_Border,TRUE,
   GTNM_MaxNumberLen,3,
   GTNM_Format,"%ld",
   TAG_DONE))) machszu(20);

/* Num2 Gadget erstellen */
```

```
   mynewgadget.ng_LeftEdge=181;
   mynewgadget.ng_GadgetText="2";
   mynewgadget.ng_GadgetID=7;
   mynewgadget.ng_Flags=PLACETEXT_BELOW;

   if(!(myNum2=CreateGadget(NUMBER_KIND,myNum1,&mynewgadget,
   GTNM_Justification,GTJ_CENTER,
   GTNM_Border,TRUE,
   GTNM_MaxNumberLen,3,
   GTNM_Format,"%ld",
   TAG_DONE))) machszu(20);

/* Num3 Gadget erstellen */

   mynewgadget.ng_LeftEdge=242;
   mynewgadget.ng_GadgetText="3";
   mynewgadget.ng_GadgetID=7;
   mynewgadget.ng_Flags=PLACETEXT_BELOW;

   if(!(myNum3=CreateGadget(NUMBER_KIND,myNum2,&mynewgadget,
   GTNM_Justification,GTJ_CENTER,
   GTNM_Border,TRUE,
   GTNM_MaxNumberLen,3,
   GTNM_Format,"%ld",
   TAG_DONE))) machszu(20);
```

Zunächst wird die VisualInfo des aktuellen PublicScreens erfragt. Gleichzeitig wird dieser gesperrt, sodaß nicht bis zum Öffnen des Windows der Screen geschlossen werden kann, was zu einem schweren Fehler führen würde.

Danach wird eine Gadgetbasis erstellt, in die nach und nach die einzelnen Gadgets mit allen Startattributen verkettet eingetragen werden. Sollte bei all diesen Betriebssystemaufrufen ein Fehler (z.B. durch Speichermangel) auftreten, wird mit einem Fehlercode für schwere Fehler das Programm beendet.

10.2.3.4 Window öffnen

```
/* Window öffnen */

   if (!( mywindow = OpenWindowTags(NULL,
      WA_InnerWidth, 310,
      WA_InnerHeight, 205,
      WA_GimmeZeroZero, TRUE,
      WA_IDCMP, CLOSEWINDOW | GADGETUP | GADGETDOWN,
      WA_SizeGadget, FALSE,
      WA_DepthGadget, TRUE,
      WA_DragBar, TRUE,
      WA_Title, "DoSwitch",
      WA_CloseGadget, TRUE,
      WA_NoCareRefresh, FALSE,
      WA_SmartRefresh, FALSE,
      WA_Activate, TRUE,
      WA_Gadgets, mygadgetbase,
      WA_PubScreen, myscreen,
      WA_ScreenTitle, "DoSwitch",
      TAG_DONE ) ))
      machszu(20);

/* Screen wieder freigeben */

      UnlockPubScreen(NULL,myscreen);
      myscreen=NULL;
```

Nun wird das Window geöffnet und der PublicScreen wieder freigegeben, da dieser nun wegen des eröffneten Windows nicht mehr geschlossen werden kann.

10.2.3.5 Ports für das serielle Device öffnen

```
/* Ports herstellen */

   if ((SerialMP = (struct MsgPort *) CreatePort(0, 0)) == NULL)
   {
      myeasy.es_TextFormat="---======* ERROR *********\nKann MessagePort nicht öffnen!";
      EasyRequest(mywindow,&myeasy,NULL);
      machszu(20);
   }
   if ((SerialIO = (struct IOExtSer *) CreateExtIO(SerialMP, sizeof(struct IOExtSer))) == NULL)
   {
      myeasy.es_TextFormat="---======* ERROR *********\nKann ExtIO nicht kreieren!";
      EasyRequest(mywindow,&myeasy,NULL);
      machszu(20);
   }
   if ((IncomingIO = (struct IOExtSer *) CreateExtIO(SerialMP, sizeof(struct IOExtSer))) == NULL)
   {
      myeasy.es_TextFormat="---======* ERROR *********\nKann ExtIO nicht kreieren!";
      EasyRequest(mywindow,&myeasy,NULL);
      machszu(20);
   }
```

Der Messageport und die beiden IOStrukturen für das serielle Device werden geöffnet; beim Fehlschlagen kann wegen des bereits geöffneten Windows mit Requestern darauf hingewiesen werden, bevor das Programm mit einem Fehlercode beendet wird.

10.2.3.6 serielles Device öffnen und konfigurieren

```
/* Device öffnen */

   if ((SerOpen = !OpenDevice("serlog.device", 0L, (struct IORequest *) SerialIO, 0L)) == NULL)
   {
      myeasy.es_TextFormat="---======* ERROR *********\nKann das Device nicht öffnen!";
      EasyRequest(mywindow,&myeasy,NULL);
      machszu(20);
   }

/* Device konfigurieren */

   SerialIO->io_Baud = 9600;
   SerialIO->io_ReadLen = 8;
   SerialIO->io_WriteLen = 8;
   SerialIO->io_StopBits = 1;
   SerialIO->io_SerFlags = (SERF_RAD_BOOGIE | SERF_XDISABLED);
   SerialIO->IOSer.io_Command = SDCMD_SETPARAMS;
   if (DoIO((struct IORequest *) SerialIO) != NULL)
   {
      myeasy.es_TextFormat="---======* ERROR *********\nKann keine Konfiguration durchführen!";
      EasyRequest(mywindow,&myeasy,NULL);
      machszu(10);
   }
   memcpy(IncomingIO, SerialIO, sizeof(struct IOExtSer));

/* Verbindung aufbauen */

   DoConnect();
```

Zuerst wird das serielle Device geöffnet und danach mit einer entsprechend initialisierten Struktur konfiguriert. Bei Fehlern wird wieder eine Meldung über Requester ausgegeben und danach das Programm mit einem entsprechenden Fehlercode beendet. Sollte alles geklappt haben, wird versucht die Verbindung zum Multiswitch aufzubauen und eine Erfolgsmeldung wird im Info–Gadget angezeigt.

10.2.3.7 Kontrollroutine

```
/* Warten für immer */

   for(;;)
   {
      Wait(1L<< mywindow->UserPort->mp_SigBit);

/* Holen und Auswerten der Message */

      myMsg=GT_GetIMsg((struct MsgPort *) mywindow->UserPort);
      GT_ReplyIMsg(myMsg);
      if (myMsg)
      {
         if (myMsg->Class==IDCMP_CLOSEWINDOW)
         {
            machszu(0);
         }
         else /* muß wohl ein Gadget gewesen sein */
         {
            mygadget=(struct Gadget *) myMsg->IAddress;
            switch(mygadget->GadgetID)
            {
               case 8: /* Read */
               case 1: /* List = Read */
                  ReadReg(1);
                  break;
               case 3: /* Write */
                  Buffer[0]=24;  /* Write-Befehl senden */
                  SWrite(1);
                  SRead(1);
                  if(Buffer[0])
                  {
                     GT_SetGadgetAttrs(myInfo,mywindow,NULL,
                     GTTX_Text,"Schreibfehler!",
                     TAG_DONE);
                     break;
                  }
                  else
                  {  /* aktuelles Register holen */
                     if(!(GT_GetGadgetAttrs(myList,mywindow,NULL,
                     GTMX_Active,&myReg,
                     TAG_DONE))) machszu(20);
                     Buffer[0]=myReg+1;
                     SWrite(1);  /* Register senden */
                     SRead(1);
                     if(Buffer[0])
                     {
                        GT_SetGadgetAttrs(myInfo,mywindow,NULL,
                        GTTX_Text,"falsches Reg!",
                        TAG_DONE);
                        break;
                     }
                     else
                     { /* Wert holen */
                        if(!(GT_GetGadgetAttrs(mySlider,mywindow,NULL,
                        GTSL_Level,&myVal,
                        TAG_DONE))) machszu(20);
                        Buffer[0]=myVal;
                        SWrite(1);  /* Wert senden */
                        SRead(1);
                        if(Buffer[0])
                        {
                           GT_SetGadgetAttrs(myInfo,mywindow,NULL,
                           GTTX_Text,"Schreibfehler!",
                           TAG_DONE);
                           break;
                        }
                        else
                        {
                           GT_SetGadgetAttrs(myInfo,mywindow,NULL,
                           GTTX_Text,"Schreiben OK",
                           TAG_DONE);
                           break;
                        }
                     }
                  }
                  break;
               case 5: /* Connect */
                  DoConnect();
                  break;
```

```
                case 9: /*Reset */
                   GadOff();      /* Bedienungselemente aus */
                   GT_SetGadgetAttrs(myConnect,mywindow,NULL,
                   GA_Disabled,TRUE,
                   TAG_DONE);
                   GT_SetGadgetAttrs(myInfo,mywindow,NULL,
                   GTTX_Text,"Reset...",
                   TAG_DONE);
                   Buffer[0]=3;   /* Reset-Befehl senden */
                   SWrite(1);
                   SRead(1);
                   if(!(Buffer[0]))
                   {
                      if(!(GT_GetGadgetAttrs(myCycle,mywindow,NULL,
                      GTCY_Active,&myReg,
                      TAG_DONE))) machszu(20);
                      if(myReg)
                      {
                         Buffer[0]=0xff;
                      }
                      else
                      {
                         Buffer[0]=0;
                      }
                      SWrite(1);
                      SRead(1);      /* Modus anerkannt? */
                      if (!(Buffer[0]))
                      {/* OK - Neuaufbau */
                         SRead(2);
                         if((Buffer[0]==0xff)&&(Buffer[1]==0))
                         {
                            Buffer[0]=0;   /* OK */
                            Buffer[1]=2;   /* eigene Kennung */
                            SWrite(2);
                            SRead(1);
                            if(!(Buffer[0]))
                            { /* Verbindung wiederhergestellt */
                               GadOn();
                               GT_SetGadgetAttrs(myConnect,mywindow,NULL,
                               GA_Disabled,TRUE,
                               TAG_DONE);
                               GT_SetGadgetAttrs(myInfo,mywindow,NULL,
                               GTTX_Text,"Reset OK",
                               TAG_DONE);
                               break;
                            }
                         }

/* Möglichkeit des manuellen Verbindungsaufbaus freigeben */

                         GT_SetGadgetAttrs(myConnect,mywindow,NULL,
                         GA_Disabled,FALSE,
                         TAG_DONE);
                         GT_SetGadgetAttrs(myInfo,mywindow,NULL,
                         GTTX_Text,"keine Verbindung!",
                         TAG_DONE);
                         break;
                         }
                         else
                         {
                         GadOn();
                         GT_SetGadgetAttrs(myConnect,mywindow,NULL,
                         GA_Disabled,TRUE,
                         TAG_DONE);
                         GT_SetGadgetAttrs(myInfo,mywindow,NULL,
                         GTTX_Text,"falscher ResMode!",
                         TAG_DONE);
                         break;
                         }
                      }
                      else
                      {
                         GadOn();
                         GT_SetGadgetAttrs(myConnect,mywindow,NULL,
                         GA_Disabled,TRUE,
                         TAG_DONE);
                         GT_SetGadgetAttrs(myInfo,mywindow,NULL,
                         GTTX_Text,"kein Reset!",
                         TAG_DONE);
                         break;
                      }
```

```
            }
          }
        } /* falscher Alarm, nur für GadTools! */
    }
}
```

Die gesamte Routine wird solange wiederholt, bis das Schließgadget des Windows betätigt wird. Zu Beginn wird der Task in den Wartezustand versetzt bis eine Message eintrifft. Diese wird vom Messageport geholt (`GT_GetIMsg`) und beantwortet (`GT_ReplyIMsg`). Dann beginnt die Untersuchung der Message: wenn es das Schließgadget war, wird das Programm beendet, wenn nicht, kann es nur ein Gadget gewesen sein (andere Messages werden ignoriert, da sie nur systemintern sein können) und die Gadget–ID wird geholt. Aufgrund der Gadget–ID wird nun für jedes Gadget eine entsprechende Aktion ausgelöst.

- Sowohl das Read–Gadget als auch das MutualExklude–Gadget führen dazu, daß das aktuelle Register aus dem Multiswitch ausgelesen und angezeigt wird.
- Das Write–Gadget beschreibt das aktuelle Register des Multiswitches mit dem im Slider–Gadget eingestellten Wert. Dazu wird zunächst der Writebefehl an den Multiswitch gesendet. Danach folgen — immer positive Antworten vorausgesetzt — die Registernummer und der Datenwert, wobei die Registernummer aus dem MutualExklude–Gadget und der Datenwert aus dem Slider–Gadget gewonnen werden. Sollte ein Fehler auftreten, so wird dieser im Info–Gadget angezeigt, sonst eine Erfolgsmeldung.
- Das Connect–Gadget baut eine Verbindung zum Multiswitch auf.
- Das Reset–Gadget löst einen Warm– oder Coldreset (je nach Stellung des Cycle–Gadgets) im Multiswitch aus. Dazu wird der Resetbefehl mit dem entsprechenden Argument an den Multiswitch gesandt. Sowohl Fehler als auch Erfolgsmeldungen werden im Info–Gadget angezeigt. Im Anschluß daran wird die Verbindung mit dem Multiswitch auf Betreiben des Multiswitches neu aufgebaut. Sollte das scheitern, werden alle Gadgets bis auf das Connect–Gadget deaktiviert und eine Fehlermeldung wird angezeigt.

10.3 die Funktionen und die Bedienung

10.3.1 Start

Das Programm wird — wie üblich — mit einem Doppelclick auf das Programmicon gestartet. Sollte vor dem Aufbau des Fensters ein schwerer Fehler auftreten, so bricht das Programm ohne sichtbare Fehlermeldung ab. In diesem Fall sind Probleme im System der

Grund dafür und ein erneuter Versuch nach einem Neustart sollte zum Erfolg führen; eine Verbesserung zukünftiger Versionen könnte in Alert–Fehlermeldungen in solchen Fällen bestehen. In dieser ersten Version wurde jedoch davon abgesehen, da solche Fehler sehr selten sind und während der Probeläufe auch kein einziger auftrat.

Wenn es Fehler nach dem Windowaufbau gibt (z.B. ein bereits belegtes serielles Device), so wird zuerst ein Requester aufgebaut (siehe Abbildung 10.1), der darauf hinweist. Nach

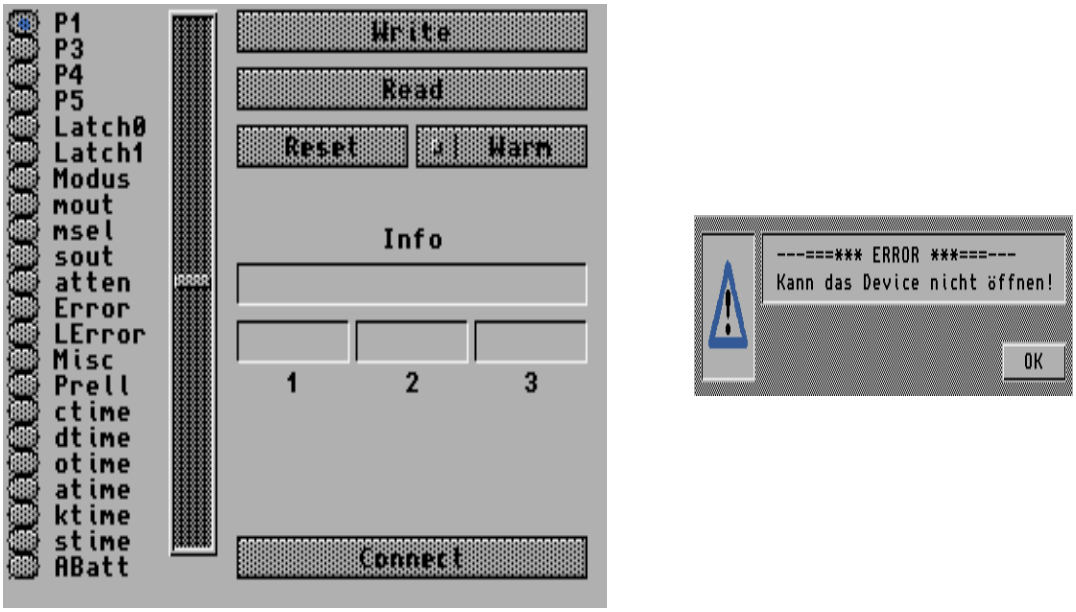

Abbildung 10.1: Programm mit Fehlermeldung durch serielles Device: links Programmfenster — rechts Fehlermeldungs–Requester

einer Bestätigung, bricht das Programm ab.

Wenn kein schwerer Fehler auftritt und die Verbindung mit dem Multiswitch aufgebaut werden kann, so werden alle Bediengadgets (außer Connect) freigegeben und man kann mit der Kommunikation mit dem Multiswitch beginnen (siehe Abbildung 10.2 links). Sollte es zu keiner Verbindung gekommen sein, so bleiben alle Gadgets (außer Connect) deaktiviert und man kann einen erneuten Versuch mittels des Gadgets Connect starten (siehe Abbildung 10.2 rechts).

10.3.2 Bedienung

Zur Bedienung stehen einem die drei Befehle *Read*, *Write* und *Reset* zur Verfügung. Sowohl der Read– als auch der Writebefehl beziehen sich immer auf das mit dem MutualExklude–Gadget eingestellte Register. Beim Auswählen eines neuen Registers wird automatisch der

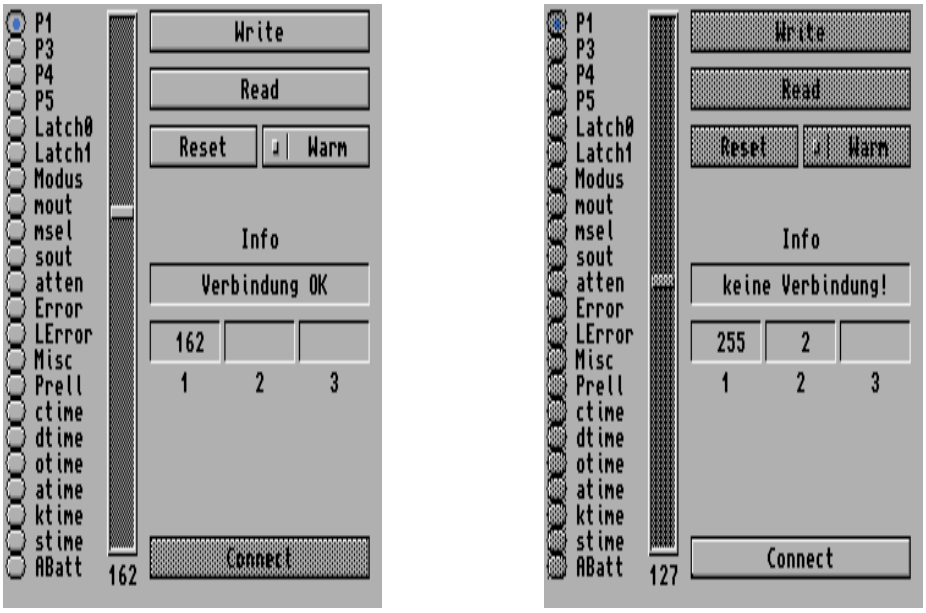

Abbildung 10.2: Programm nach dem Start: links erfolgreicher Verbindungsaufbau — rechts erfolgloser Verbindungsaufbau

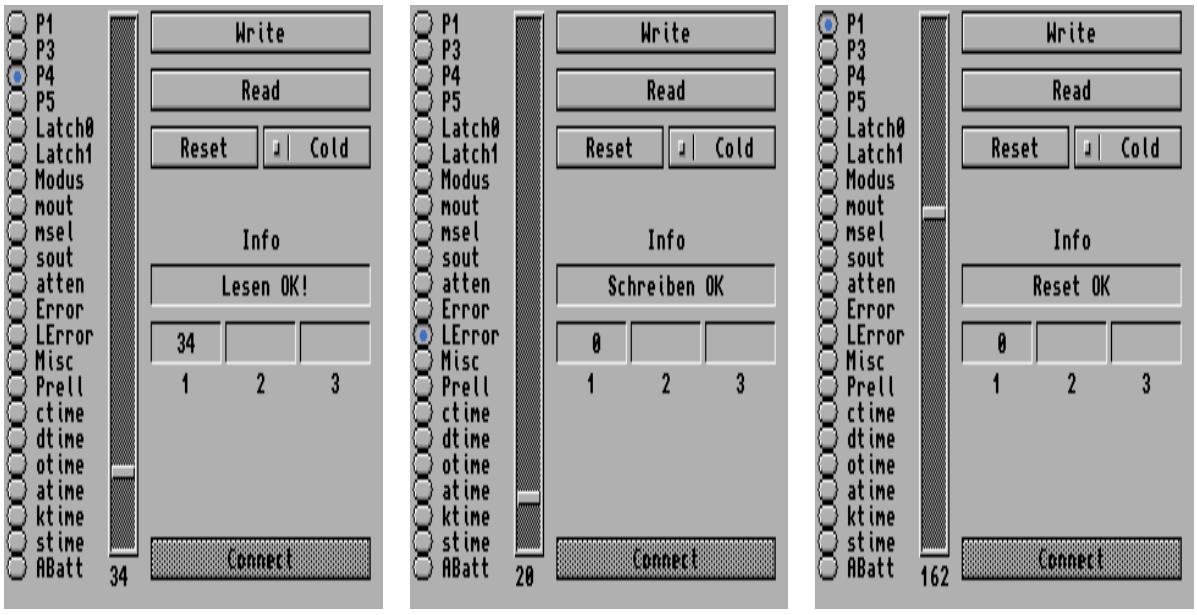

Abbildung 10.3: Programm nach den drei Operationen: links nach Read — mitte nach Write — rechts nach Reset

aktuelle Wert aus dem Multiswitch gelesen.

Read: Hiermit wird das eingestellte Register erneut ausgelesen und angezeigt.

Write: Mit diesem Gadget wird der im Slider–Gadget eingestellte Wert im aktuellen Register gespeichert.

Reset: Dadurch kann im Multiswitch ein Warm– oder Coldreset — je nach Einstellung des Cycle–Gadgets — durchgeführt werden.

Bei allen Operationen werden die Rückgabewerte ohne Vorbearbeitung in den drei Nummerngadgets angezeigt, sodaß Fehlinterpretationen des Programms ausgeschlossen werden können. Weiters werden die aktuellen Informationen über den Ausgang des letzten Befehls im Info–Gadget dargestellt, sodaß man immer weiß, ob ein Befehl erfolgreich war, oder warum er nicht erfolgreich war. Da das Protokoll jeden Einzelschritt der Befehlsübermittlung einzeln bestätigt, kann auch eine sehr detaillierte Fehlermeldung ausgegeben werden. Daher kann z.B. bei einem Schreibbefehl zwischen nicht angenommenen Schreibbefehl, falschem Register (z.B. P5) und sonstigen Problemen unterschieden werden.

10.3.3 Ende

Das Programm kann durch Clicken auf das Standardschließgadget des Windows beendet werden.

10.4 mögliche Erweiterungen

Dieses Programm stellt eine erste Version für mich als Entwickler dar. Weitere Versionen sollten zunächst nicht alle Register beschreiben können; eine Anordnung in mehreren Ebenen (siehe Kapitel 8.3.3.2) ist zu empfehlen. Weiters wäre — wie oben erläutert — eine erweiterte Fehleranzeige beim Programm wünschenswert. Als nächsten Schritt könnte man eine bessere Anzeige mancher Register verwirklichen. So sollte man die Batteriespannung direkt in Volt angeben, oder das Errorregister gleich auswerten und die entsprechenden Fehlermeldungen anzeigen. Zu allerletzt wäre eine bessere GUI (fontsensitiv, übersichtlicher, ...) sicher sinnvoll.

Kapitel 11

Entwicklung der Hardware des Multiswitches

Dieses Kapitel zeigt die Entwicklung der Hardware, wie sie in der letzten Version der Diplomarbeit hergestellt wurde. Diese Version ist nicht identisch mit dem ursprünglichen Aufbau, da erst anhand des ersten Aufbaus Fehler entdeckt und ausgebessert werden konnten. Ich erläutere aber aus Gründen der Übersichtlichkeit hier die „komplette“ Version und erst im Kapitel 13.2.2 die Fehlersuche der Urversion.

11.1 Übersicht

Die Hardware ist aus verständlichen Gründen sehr einfach, da fast alle Operationen vom Prozessor übernommen werden, sodaß die meiste Hardware für erweiterte Ein/Ausgabemöglichkeiten und Anpassungen von Hardware an den Prozessor besteht. In der Abbildung 11.1 ist der Gesamtschaltplan des Prints zu sehen. In den folgenden Kapiteln werden nun Teilbereiche daraus erläutert und bei Bedarf dimensioniert.

11.2 Versorgung

Die Versorgung soll — nach den Definitionen aus dem Kapitel 4.6 — extern erfolgen; ein Schutz gegen Verpolung und in bestimmten Maße gegen Überspannung war vorzusehen. Aus diesem Grund habe ich den Eingang gleich nach dem Stecker mit einer Sicherung geschützt (Überstromsicherung); daran anschließend folgt ein Gleichrichter zur Verpolungssicherung. Die Überspannungssicherung wird durch den verwendeten Regler 7805 (IC1) gewährleistet. Vor den Regler habe ich eine Kombination aus Entstörkondensator (C2 =

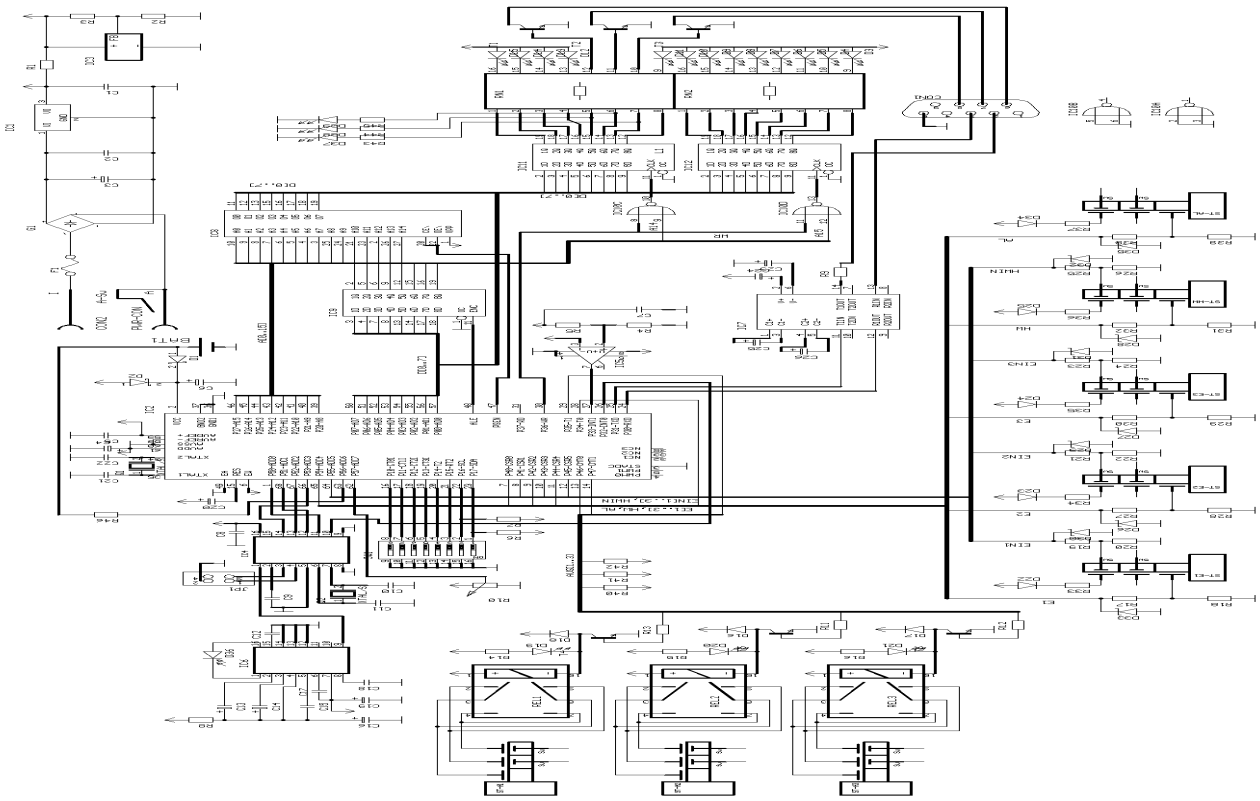

Abbildung 11.1: Gesamtschaltplan des Prints

100nF / 63V) und Ladekondensator (C3 = 470μF / 50V) gesetzt; die hohen Spannungswerte wurde wegen der Eingangspannungssicherheit gewählt. Am Ausgang des Reglers habe ich wie üblich einen Entstörkondensator (100nF) angebracht.

Weiters benötigt der Print für den ADC im Prozessor eine Referenzspannung. Diese wurde zu 3.5V gewählt, damit auch neue Lithiumbatterien mit Spannungen größer 3V oder 3.3V Batterien noch sicher gemessen werden können. Erreicht wird diese durch die „programmierbare“ Spannungsreferenz LM385 von National. Für den Widerstand R3[1] habe ich 100kΩ gewählt. Daher ergibt sich für den zweiten Widerstand (R2) mit der Berechnungsformel aus Abbildung 11.2 ein Wert von 180kΩ (bereits auf E24 gerundet). Durch die Rundung ergibt sich eine reale Referenzspannung von 3.47V. Somit kann durch einen 8–Bit–ADC eine Auflösung von 13.6mV erreicht werden, was für eine Batteriemessung ausreichen sollte.

Der Widerstand R1 dient zur Querstromeinstellung. Um die Referenzspannung, die von drei Bauteilen verwendet wird, stabil genug zu machen, habe ich einen Querstrom von

[1] nicht des Beispiels, sondern aus dem Gesamtschaltplan 11.1!

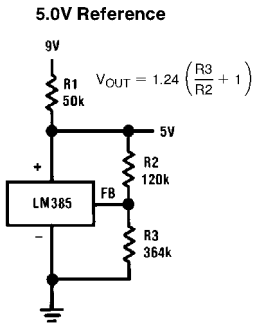

Abbildung 11.2: Beispielschaltung für die Dimensionierung der Referenzspannungsquelle [2]

1mA gewählt. Aus dem Ohmschen Gesetz ergibt sich (nach Rundung auf die E24 Reihe) ein Widerstand von 1.5kΩ.

11.3 IR–Empfänger

Ich habe die zwei IR–Empfängerbausteine von Plessey verwendet, da die EOG–Brille den entsprechenden Sendebaustein verwendet. IC6 ist der Empfänger– und Verstärkerbaustein (SL486), IC4 (MV601) ist der Konverter von PPM auf PCM mit einem μP–Port. Die Schaltung habe ich direkt aus den Datenblättern [3] übernommen, sodaß keine weitere Erläuterung notwendig ist.

Einzig die beiden Jumper bedürfen einer Erläuterung. Der zugehörige Sendebaustein MV500 kennt drei Betriebsebenen. Da zum Zeitpunkt des Printentwurfes nicht sicher gestellt war, daß nur eine einzige Ebene für die Sender verwendet wird, wurde die Empfängerschaltung mit den Jumpern konfigurierbar gemacht. Die drei Moden können nun durch Einstecken eines Jumpers an Pin 3 oder 4 bzw. durch Leerlassen eingestellt werden, beide Jumper gleichzeitig dürfen nie eingesteckt werden, da dies keine zulässige Ebene auswählt.

Der Anschluß an den 80C552 gestaltet sich unproblematisch, da die vier Bitleitungen direkt mit dem Port 5 verbunden werden und ein neuer Code mittels Interrupt gemeldet wird.

11.4 der Prozessor

Der Prozessor (IC2, 80C552 oder 80C562) selbst benötigt wenige externe Bauteile nötig. Die Versorgung wird über zwei Schottky–Dioden (D1 und D2) geführt, um den Spannungs-

verlust so gering wie möglich zu halten. Dies war notwendig, da der Prozessor keinen eigenen Spannungseingang für den RAM besitzt. Daher wird die Versorgung bei ausgefallener externer Spannung automatisch von der Batterie übernommen. Da — wie im Kapitel 11.8 gezeigt — eine Mindestzeit vergehen muß, um den externen Spannungsverlust zu detektieren und den Prozessor in den Power–Down–Modus überzuführen, wurde ein entsprechend großer Ladekondensator direkt am Prozessor eingeführt. Wie sofort einsichtig ist, ergibt sich der Wert des Kondensators aus folgender Formel:

$$C = \frac{t}{-125 \ln(0.9)} \tag{11.1}$$

Dabei wurde bereits der geschätzte Verbraucherwiderstand von 125Ω und ein erlaubter Spannungsabfall auf 90% in der Formel berücksichtigt. Um sicher zu sein, daß der Prozessor aus jeder Situation (auch während einer ADC–Konversion) in den Power–Down–Modus kommt, habe ich eine Mindestzeit von 4ms gewählt. Aus der Formel 11.1 ergibt sich (nach runden auf handlsübliche Werte) ein Kondensator von 330μF.

Der Quarzresonator wurde mit einer Frequenz von 11.0592MHz gewählt, um eine günstige Timer–Reload–Value für die Baudratengeneration zu erhalten.

Der Reset wird durch den internen Widerstand des Prozessors und dem empfohlenen externen Kondensator von 2.2μF verwirklicht.

Die Referenzspannungsversorgung wird durch die im Kapitel 11.2 beschriebene Referenzspannungsquelle realisiert.

Zuletzt wurden noch die Eingänge $\overline{\text{EA}}$ (kein interner ROM) und STADC (kein externer ADC–Start) auf GND gelegt.

11.5 ROM

Da dieser Print einen experimentellen Charakter hat, wurde der ROM als externer EPROM ausgelegt. Dadurch gehen 2 Ports für die Daten und Adressleitungen verloren; aus diesem Grund mußten zwei externe Latches verwendet werden.

Da am Port 0 die Daten und die untere Häfte des Adressbusses gemultiplext übertragen werden, benötigt diese Schaltungsvariante einen Datenspeicher für den unteren Adressbusteil. Dieses Latch wurd durch den IC9 (74HC373) verwirklicht, die Steuerung erfolgt durch die ALE–Leitung.

Als ROM habe ich den derzeit günstigsten EPROM (IC8; 27C256) gewählt, der auch Programmerweiterungen sicher aufnehmen kann. Durch die Verwendung eines externen ROMs, konnte auch bei der Fehlersuche ein EPROM–Simulator eingesetzt werden, der diesen Vorgang deutlich beschleunigte.

11.6 externe Latches

Wie im vorigen Kapitel erläutert, mußten für diese Version des Prints zwei externe Latches verwendet werden. Diese habe ich Memorymapped ins System eingebunden. Da die $\overline{\text{WR}}$–Leitung low–aktiv ist, habe ich die Adressen 14 und 15 zur Auswahl der beiden Latches ebenfalls low–aktiv gewählt[2], um mit einem NOR (IC10) eine Und–Verknüpfung erreichen zu können. Als Latches wurden die Bausteine 74HC574 (IC11 und IC12) verwendet. Die Bausteine aus der HC–Reihe sind in der Lage, sowohl Strom nach GND aufzunehmen als auch Strom von VCC zu liefern. Daher konnten die high–aktiven LEDs für die Mousetasten direkt (über Serienwiderstände) an die Latches geschaltet werden.

11.7 LEDs und Transistoren an den Latches

Neben den im vorigen Kapitel angesprochenen drei LEDs für die Mousetasten (D37–D39), sind noch 12 weitere LEDs low–aktiv an einem Widerstandsnetzwerk (RN1 und RN2) über die Latches ansprechbar. Der Wert für die Vorwiderstände ergibt sich aus dem Ohmschen Gesetz, der Versorgungsspannung von 5V, der Flußspannung der LEDs von ca. 2V und dem typischen Strom von 2mA (nach Rundung auf handelsübliche Werte) zu 1kΩ. Gleichzeitig sind an diesem Widerstandsnetzwerk die 3 Transistoren (T1–T3) für die Mousetastenemulation angeschlossen, für die ich Standardtransistoren der Type BC337 (500mA maximaler Kollektorstrom) gewählt habe.

11.8 Versorgungsspannungsüberwachung

Zur Spannungsüberwachung habe ich einen externen Komparator verwendet, da dieser schnell und programmunabhängig einen Interrupt (mit hoher Priorität) auslösen kann. Ich habe für diesen Zweck den Standardkomparator LM311 von National verwendet. Die Beschaltung ist denkbar einfach und besteht aus zwei Widerständen, die ich als Spannungsteiler der Versorgungsspannung verwende. Die Schaltschwelle habe ich mit 4.6V festgelegt. Nach Wahl des Widerstandes R4 zu 100kΩ ergibt sich der Widerstand R5 aus dem Ohmschen Gesetz (und nach Runden auf die Reihe E24) zu 33kΩ; daraus resultiert eine wahre Schaltschwelle von 4.62V. Zur Unterdrückung von Spikes, die eine Fehlauslösung verursachen könnten, habe ich zum Spannungsteiler einen Kondensator (100nF) parallel geschaltet.

[2] siehe Kapitel 9.7.1.6

11.9 serielle Treiber/Empfänger

Zur Unterstützung einer RS232–Schnittstelle nach den aktuellen Normen habe ich den sehr weitverbreiteten IC MAX232 (IC7) verwendet, der die notwendige Spannungsverdopplung und –Invertierung aus den 5V ebenfalls intern erledigt. Die Beschaltung mit den 10μF Kondensatoren erfolgte konform zum Datenblatt [7]. Der Widerstand R9 mit einem Wert von 220Ω zum Transmitterausgang wurde zum Schutz gegen Kurzschlüsse eingeführt, da der IC selbst dafür keine Schutzmaßnahmen vorsieht.

11.10 DIP–Switches

Die DIP–Switches wurden einerseits direkt mit dem Prozessor verbunden und andererseits mit GND; pullup–Widerstände werden nicht benötigt, da diese — wie im Kapitel 9.6.1 beschrieben — bereits im Prozessor enthalten sind.

11.11 Analogeingänge

Es sind zwei Analogeingänge für den Prozessor verwirklicht worden. Zum einen habe ich einen Trimmer direkt zwischen der Referenzspannung und GND geschaltet, wobei der Abgriff mit dem Port 5 des Prozessors verbunden ist. Um die Belastung der Referenzquelle niedrig zu halten, eine Verfälschung des Meßergebnisses durch sie Belastung des Trimmers aber ebenfalls nicht zu groß werden zu lassen, habe ich einen Wert von 100kΩ gewählt.

Zum anderen wurde die Batterie über einen Widerstand von 270kΩ mit dem Port 5 verbunden. Der Serienwiderstand ist notwendig, da ansonsten im Power–Down–Modus die Batterie mit einigen Milliampere belastet werden würde; ein höherer Wert des Widerstandes führt zu erheblichen Meßfehlern.

11.12 Relaisausgänge

Die drei Relais werden mittels Transistoren (T4–T6) angesteuert, da ein Strom von ca. 30mA für die Prozessorports zu hoch ist; als Treibertransistoren wurden die Standardtransistoren BC237 (100mA maximaler Kollektorstrom) verwendet. Parallel zu den Relais wurden Freilaufdioden (1N4148) und LEDs mit Vorwiderständen geschaltet. Die Ansteuerung der Transistoren erfolgt über Längs– und Pullup–Widerstände, da zum Zeitpunkt des Umschaltens die starken Transistoren des Ports den Längswiderstand benötigen, zum Erhalten

des Ausgangsstromes (keine starken Pullups) aber externe Pullup–Widerstände benötigt werden.

11.13 Eingänge

Alle Eingänge über Klinkenbuchsen müssen nicht nur mit dem Prozessor verbunden werden, sondern sind auch gegen Überspannung zu sichern. Weiters habe ich Buchsen mit Schaltkontakten verwendet, um erkennen zu können, ob ein Stecker eingesteckt ist oder nicht.

Pro Buchse (außer der Alarmbuchse) werden zwei Leitungen zum Prozessor geführt, die alle auf die gleiche Weise geschützt sind; ein Serienwiderstand von 470Ω und eine Ableitzenerdiode mit einer Schwellspannung von 5.6V verhindern Über– und Unterspannungen wirkungsvoll.

Um den Schaltzustand der Sensoren überprüfen zu können, wird an die Buchse über einen Serienwiderstand von 220Ω und einer Seriendiode (1N4148) die Versorgungsspannung angelegt. Da es sinnvoll erscheint, daß sich ein nicht angesteckter Sensor wie ein nicht aktivierter Sensor verhält, ist es notwendig, den Prozessorport passiv mittels eines Widerstandes nach Low zu ziehen und über die Taste nach High zu schalten. Da diese passiven Widerstände im Fall eines aktivierten Sensors zu einem erhöhten Stromverbrauch führen, sollten diese Widerstände möglichst groß sein. Eine obere Grenze wird jedoch durch den internen Pullup–Transistor des Ports gesetzt. Nach Rückfragen beim Hersteller des Prozessors (Philips) stellte sich heraus, daß ein Widerstand von 2.7kΩ die obere Grenze für ein sicheres Schaltverhalten darstellt. Da jedoch auch die Schutzserienwiderstände zu berücksichtigen sind, habe ich die Pulldown–Widerstände mit 2.2kΩ gewählt.

11.14 Stückliste

Aus den vorangegangenen Kapiteln folgt nun die Stückliste für den Print in den Tabellen 11.1 bis 11.3.

IC1	L7805	TO220	1A
IC2	80C552	PLCC68	
Sockel		PLCC68	
IC3	LM385Z	TO92	
IC4	MV601	DIL16	
IC5	LM311N	DIL8	
IC6	SL486	DIL16	
IC7	MAX232	DIL16	
IC8	M27C256	DIL28	
Sockel		DIL28	
IC9	74HC373	DIL20	
IC10	74HC02	DIL14	
IC11	74HC574	DIL20	
IC12	74HC574	DIL20	
SW1	DIL-Sw.	8pol	
JP1	2×2		
Jumper	2 St.	sw.	

Rel1	2 Umsch.	5V	140mW
Rel2	2 Umsch.	5V	140mW
Rel3	2 Umsch.	5V	140mW
Bat1	ø24×3		
	CR2450	3V	500mA
ST-A1	stereo	Klinke	
ST-A2	stereo	Klinke	
ST-A3	stereo	Klinke	
ST-E1	stereo	Klinke	
ST-E2	stereo	Klinke	
ST-E3	stereo	Klinke	
ST-HW	stereo	Klinke	
ST-AL	stereo	Klinke	
CON1	Sub-D9	90°	F
CON2	2.5mm		
F1	20mm		
Sich	flink	250mA	20mm

Tabelle 11.1: ICs, Schalter, Buchsen, Jumper, Sicherungen

R1	1k5	K	0.125W
R2	180k	M	0.125W
R3	100k	M	0.125W
R4	100k	M	0.125W
R5	33k	M	0.125W
R6	100k	K	0.125W
R7	100k	K	0.125W
R8	47E	K	0.125W
R9	220E	K	0.25W
R10	100k	Trimmer	0.15W
R11	4k7	K	0.125W
R12	4k7	K	0.125W
R13	4k7	K	0.125W
R14	1k	K	0.125W
R15	1k	K	0.125W
R16	1k	K	0.125W
R17	470E	K	0.25W
R18	2k2	K	0.25W
R19	470E	K	0.25W
R20	2k2	K	0.25W
R21	470E	K	0.25W
R22	2k2	K	0.25W
R23	470E	K	0.25W
R24	2k2	K	0.25W

R25	470E	K	0.25W
R26	2k2	K	0.25W
R27	470E	K	0.25W
R28	2k2	K	0.25W
R29	2k2	K	0.25W
R30	470E	K	0.25W
R31	2k2	K	0.25W
R32	470E	K	0.25W
R33	220E	K	0.25W
R34	220E	K	0.25W
R35	220E	K	0.25W
R36	220E	K	0.25W
R37	220E	K	0.25W
R38	470E	K	0.25W
R39	2k2	K	0.25W
R40	4k7	K	0.125W
R41	4k7	K	0.125W
R42	4k7	K	0.125W
R43	1k	K	0.125W
R44	1k	K	0.125W
R45	1k	K	0.125W
RN1	1k	M	0.25W
RN2	1k	M	0.25W

Tabelle 11.2: Widerstände

C1	100n	Polyester	63V
C2	100n	Polyester	63V
C3	470μ	Elko	50V
C4	2μ2	Tantal	35V
C5	100n	Polyester	63V
C6	330μ	Elko	6.3V
C7	100n	Polyester	63V
C8	470n	Polyester	63V
C9	100n	Polyester	63V
C10	56p	Keramik	100V
C11	56p	Keramik	100V
C12	15n	Polyester	63V
C13	6μ8	Tantal	6.3V
C14	68μ	Tantal	16V
C15	22n	Polyester	63V
C16	10μ	Tantal	16V
C17	4n7	Polyester	63V
C18	150n	Polyester	63V
C19	22μ	Elko	6.3V
C20	2μ2	Tantal	16V
C21	22p	Keramik	100V
C22	22p	Keramik	100V
C23	10μ	Tantal	16V
C24	10μ	Tantal	16V
C25	10μ	Tantal	16V
C26	10μ	Tantal	16V
Q1	11.0592MHz	AT-Quarz	
Q2	500kHz	Keramik	
G1	1.5A	400V	
T1	BC337	NPN	500mA
T2	BC337	NPN	500mA
T3	BC337	NPN	500mA
T4	BC237	NPN	100mA
T5	BC237	NPN	100mA
T6	BC237	NPN	100mA
D1	1N5817RL	Schottky	1A
D2	1N5817RL	Schottky	1A
D3	LED	gelb	3mm
D4	LED	gelb	3mm
D5	LED	gelb	3mm
D6	LED	gelb	3mm
D7	LED	grün	3mm
D8	LED	grün	3mm
D9	LED	grün	3mm
D10	LED	grün	3mm
D11	LED	rot	3mm
D12	LED	gelb	3mm
D13	LED	gelb	3mm
D14	LED	gelb	3mm
D15	LED	gelb	3mm
D16	1N4148	75mA	
D17	1N4148	75mA	
D18	1N4148	75mA	
D19	LED	rot	3mm
D20	LED	rot	3mm
D21	LED	rot	3mm
D22	1N4148	75mA	
D23	1N4148	75mA	
D24	1N4148	75mA	
D25	1N4148	75mA	
D26	BZX79	5.6V	0.5W
D27	BZX79	5.6V	0.5W
D28	BZX79	5.6V	0.5W
D29	BZX79	5.6V	0.5W
D30	BZX79	5.6V	0.5W
D31	BZX79	5.6V	0.5W
D32	BZX79	5.6V	0.5W
D33	BZX79	5.6V	0.5W
D34	1N4148	75mA	
D35	BZX79	5.6V	0.5W
D36	HP307R2	PIN	
D37	LED	rot	3mm
D38	LED	rot	3mm
D39	LED	rot	3mm

Tabelle 11.3: Kondensatoren, Resonatoren, Gleichrichter, Transistoren, Dioden

Teil III

praktische Ausführung

Kapitel 12

Printentwicklung des Multiswitches

In diesem Kapitel werden alle Unterlagen und Informationen zur Printerstellung vermittelt. Wie in den vorangegangenen Kapiteln wird die Endversion dargestellt; die Unterschiede zur Urversion und die Fehlersuche sind im Kapitel 13.2.2 zu finden.

12.1 Printerstellung

Der Print wurde auf dem CAD–Programm Eagle 3.51 DOS erstellt. Da ich zwar mit CAD–Programmen vertraut war, aber dieses Programm noch nie verwendet hatte, mußte ich erst im Rahmen von einigen kleinen Testprints die Funktionen und Librarys erkunden. Ein zusätzliches Hindernis lag in der fehlenden Beschreibung oder Kurzeinführung. Die einzige Hilfe konnte ich der einfach gehaltenen Online–Help und einigen Beispielprints und Librarys entnehmen. Daher habe ich nach dem Try&Error–Prinzip die Grundlagen der Printerstellung, Librarydefinition und Routfunktionen erlernt. Nach diesem zeitaufwendigen Lernprozeß konnte ich an meinen Diplomarbeitsprint herangehen.

Wie nicht anders zu erwarten waren nicht alle notwendigen Bauteile in den umfangreichen Librarys zu finden. Daher mußte ich parallel zur Definition des Schaltplans einige Bauteile (z.B. Schottky–Diode, Batteriehalter, Relais, Referenzspannungsquelle, Klinkenbuchse, ...) in eigenen Libraries neu definieren.

Nach langer Arbeit konnte ich an das Plazieren der Bauteile gehen. Aufgrund der Vielzahl und Größe der Bauteile entschied ich mich für einen Europrint (100×160mm) in doppelseitiger Ausführung. Nach zwei weiteren Tagen lagen alle Bauteile zumindest so günstig, daß ich auf ein recht positives Autoroutergebnis hoffen durfte.

Der Autorouter zeigte überraschenderweise nach ca. 30 Minuten Arbeit 100% an, sodaß ich für diesen ersten Probeprint nichts mehr per Hand verändern mußte.

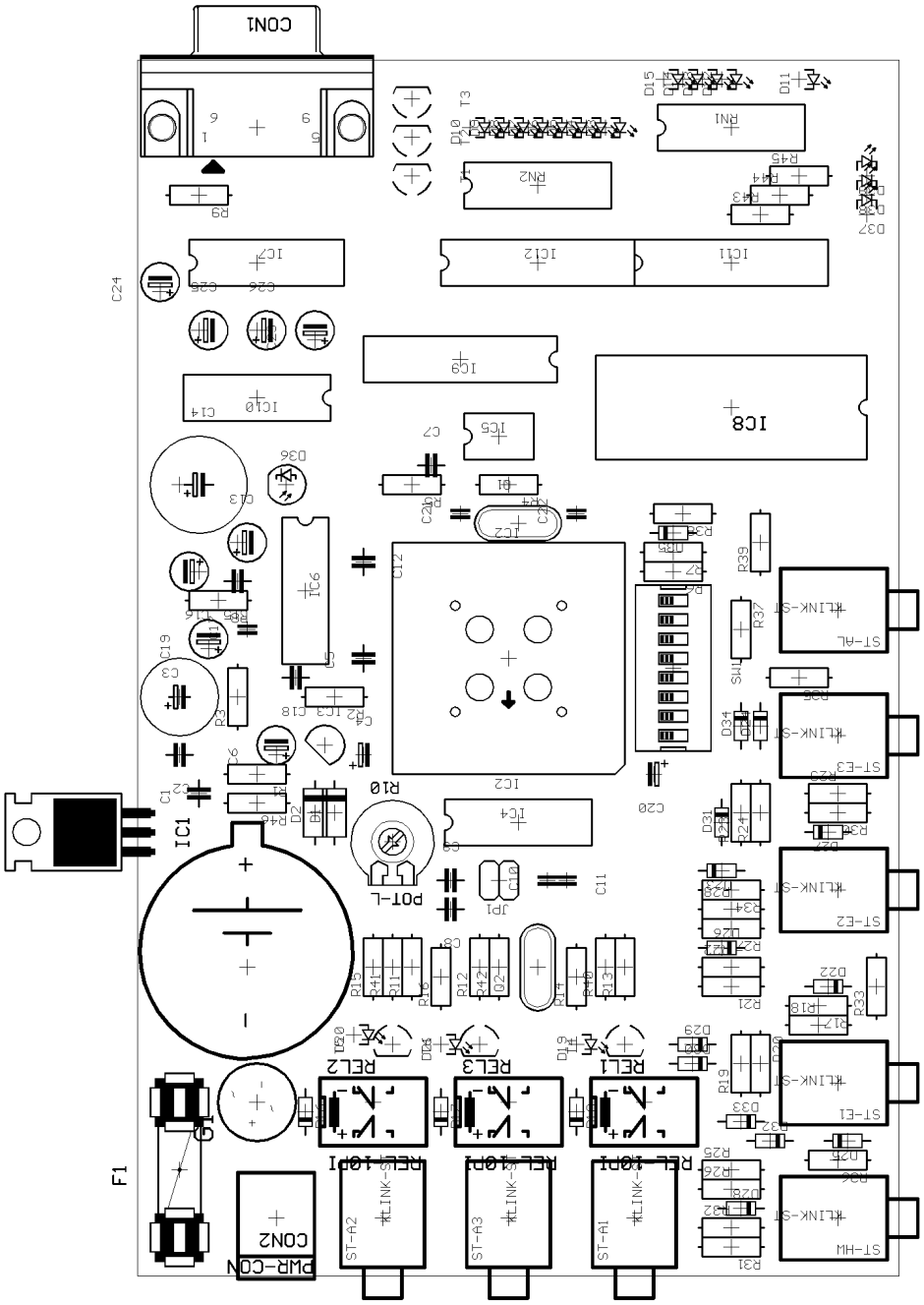

Abbildung 12.1: Bestückungsplan der Endversion

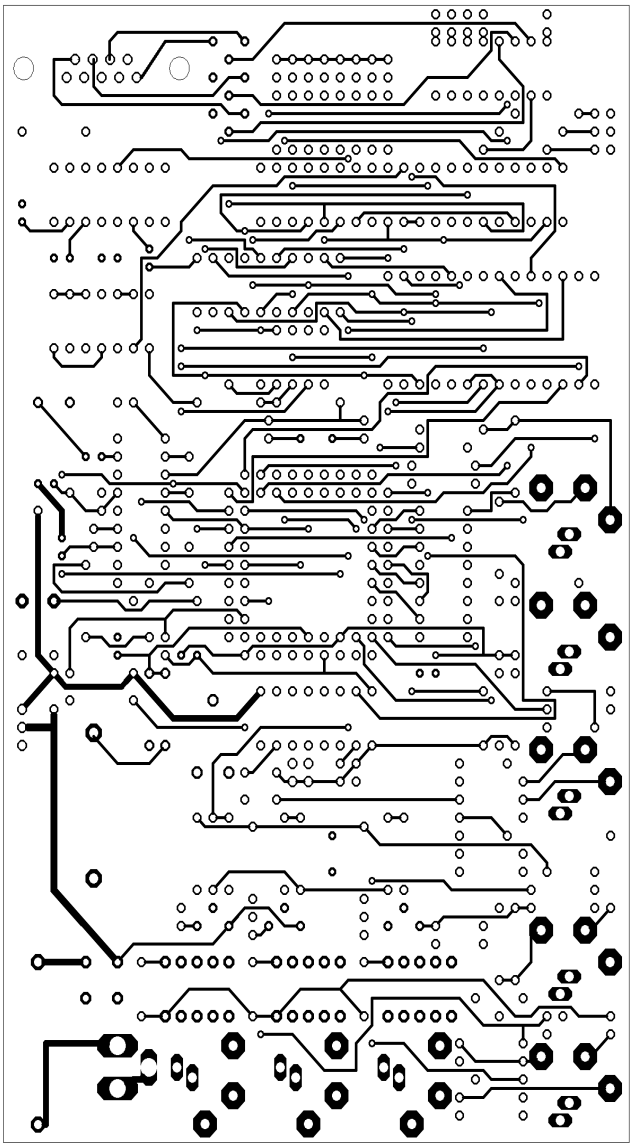

Abbildung 12.2: Bauteilseite der Endversion

12.2 Printunterlagen

Im folgenden werden die Printunterlagen im Maßstab 1:1 immer in Draufsicht[1] wiedergegeben, die nach einigen Änderungen und Handedition entstanden sind und der Endversion entsprechen. Die hier abgedruckten Unterlagen sollen auf keinen Fall zur Reproduktion des

[1] Daher ist die Lötseite zur Bauteilseite spiegelverkehrt!

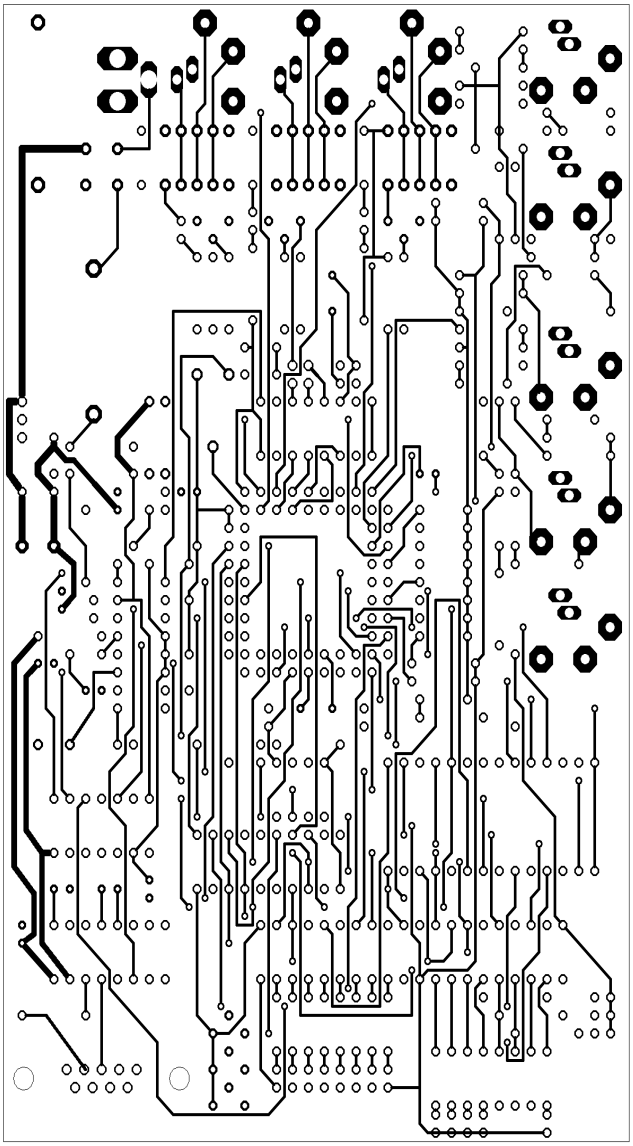

Abbildung 12.3: Lötseite der Endversion

Prints dienen, da die CAD–Unterlagen für solche Zwecke auf Datenträger zur Verfügung stehen. Die Beschriftung des Bestückungsplans ist leider etwas schwer zu lesen bzw. ist die Zuordnung zu den Bauteilen z.T. kaum nachvollziehbar, aber die Beschriftung wird in den Bauteilbibliotheken des CAD–Programms definiert, sodaß ich da kaum Abhilfe schaffen kann ohne alle verwendeten Bauteile umzudefinieren.

Kapitel 13

Herstellung der Hardware

In diesem Kapitel soll die komplette Herstellung der Hardware beschrieben werden; weiters wird auch der Fehlersuchvorgang mittels speziell dafür entwickelter Software erläutert.

13.1 Printaufbau

Der Print wurde nach den per Modem übertragenen CAD–Daten von einer Fremdfirma in doppelseitiger verzinnter Ausführung hergestellt.

Danach konnte ich an die Bestückung des Prints gehen, wobei ich bei einigen wenigen Bohrungen feststellen mußte, daß die Bohrdurchmesser der Library leider völlig an realen Bauteilen vorbeigehen. In diesen Fällen mußten entweder die Bauteilanschlüsse schmäler gefeilt, oder die Löcher aufgebohrt werden. Bei aufgebohrten Löchern mußte beim Löten darauf geachtet werden, daß der Kontakt beidseitig hergestellt wird. Trotz dieser „Widrigkeiten“ konnte der Print bald in die Probephase übergehen.

13.2 Hardwartests

Da bei einem Projekt wie diesem sowohl bei der Hardware als auch — in noch größerem Maße — bei der Software Fehler zu erwarten sind, habe ich versucht, die beiden Debug–Vorgänge zu trennen. Daher habe ich ein sehr kurzes aber dafür sehr einfaches Testprogramm geschrieben, das nur den Zweck hatte, die komplette Hardware zu testen. Nach erfolgtem Komplettest der Hardware konnte man bei Problemen mit der Anwendersoftware ziemlich sicher sein, daß ein Softwarefehler und kein Hardwarefehler daran schuld ist.

13.2.1 das Testprogramm „TheTest“

Das folgende Programm wurde für fast alle Hardwaretests herangezogen. Um den Prüfvorgang nachvollziehen zu können, wurde es hier angeführt:

```
$TITLE(TheTest)
$VERSION($VER 1.00)
$DATE(April 97)
$NODEBUG
$NOPAGING
$MOD552


 ; Stack
 ; =====

        ISEG AT 128
Stack:  DS 128

 ; Byte Direktadressen (mit Bitadressen)
 ; =====================================

        DSEG AT 32

Latch0: DS 1            ; Ausgabelatch 0
MLSel   BIT Latch0.0    ; Mouse L Sel
MMSel   BIT Latch0.1    ; Mouse M Sel
MRSel   BIT Latch0.2    ; Mouse R Sel
MOff    BIT Latch0.3    ; Mouse Off
MC      BIT Latch0.4    ; Mouse Click
MDC     BIT Latch0.5    ; Mouse D-Click
MS      BIT Latch0.6    ; Mouse Switch
Next    BIT Latch0.7    ; Next

Latch1: DS 1            ; Ausgabelatch 1
A1Sel   BIT Latch1.0    ; A1 Sel
A2Sel   BIT Latch1.1    ; A2 Sel
A3Sel   BIT Latch1.2    ; A3 Sel
MOn     BIT Latch1.3    ; Mouse On
MLT     BIT Latch1.4    ; Mouse L Taste
MMT     BIT Latch1.5    ; Mouse M Taste
MRT     BIT Latch1.6    ; Mouse R Taste
LBatt   BIT Latch1.7    ; low Batt

Port5:  DS 1            ; bitadressierbarer Speicher für P5
IR0     BIT Port5.0     ; IR D0
IR1     BIT Port5.1     ; IR D1
IR2     BIT Port5.2     ; IR D2
IR3     BIT Port5.3     ; IR D3
HM      BIT Port5.4     ; Hardware Mouse
HMin    BIT Port5.5     ; Hardware Mouse in

Out:    DS 1

 ; Byte Direktadressen (ohne Bitadressen)
 ; ======================================

        DSEG AT 48

ftime:  DS 1            ; Speicher zur Zeitmessung für's Blinken

 ; externe Byteadressen (für Latches)
 ; ==================================

        XSEG AT 0

XL0     XDATA NOT 1000000000000000b     ; Latch 0 (A15)
XL1     XDATA NOT 0100000000000000b     ; Latch 1 (A14)

 ; Bitaliases (keine Definitionen)
 ; ===============================

        BSEG

DIP     EQU P1

Al      BIT P3.4        ; Alarm
A3      BIT P3.5        ; Ausgang 3
```

```
E1      BIT P4.0        ; Eingang 1
E1in    BIT P4.1        ; Eingang 1 in
E2      BIT P4.2        ; Eingang 2
E2in    BIT P4.3        ; Eingang 2 in
E3      BIT P4.4        ; Eingang 3
E3in    BIT P4.5        ; Eingang 3 in
A1      BIT P4.6        ; Ausgang 1
A2      BIT P4.7        ; Ausgang 2


; Sonder-Zeiten

flash   EQU 42          ; Blinkzeit (ca. 0.5s)

; Basis

Bit0    EQU 00000001b
Bit1    EQU 00000010b
Bit2    EQU 00000100b
Bit3    EQU 00001000b
Bit4    EQU 00010000b
Bit5    EQU 00100000b
Bit6    EQU 01000000b
Bit7    EQU 10000000b

nBit0   EQU NOT Bit0
nBit1   EQU NOT Bit1
nBit2   EQU NOT Bit2
nBit3   EQU NOT Bit3
nBit4   EQU NOT Bit4
nBit5   EQU NOT Bit5
nBit6   EQU NOT Bit6
nBit7   EQU NOT Bit7

 ; Interrupt Startadressen
 ; =======================

        CSEG

Reset   CODE 0000H      ; Reset
X0      CODE 0003H      ; External interrupt 0
T00     CODE 000BH      ; Timer 0 overflow
X1      CODE 0013H      ; External interrupt 1
T01     CODE 001BH      ; Timer 1 overflow
SIO0    CODE 0023H      ; SIO0 (UART)
SIO1    CODE 002BH      ; SIO1 (I2C)
CT0     CODE 0033H      ; T2 capture 0
CT1     CODE 003BH      ; T2 capture 1
CT2     CODE 0043H      ; T2 capture 2
CT3     CODE 004BH      ; T2 capture 3
ADC     CODE 0053H      ; ADC completion
CM0     CODE 005BH      ; T2 compare 0
CM1     CODE 0063H      ; T2 compare 1
CM2     CODE 006BH      ; T2 compare 2
T02     CODE 0073H      ; T2 overflow

 ; Reset + Interrupts
 ; ==================

        CSEG AT Reset   ; Reset

        ajmp Init       ; zum Initialisierungscode

 ; ---------------------

        CSEG AT X0      ; IR-Ready

        ajmp Int00      ; zur Serviceroutine

 ; ---------------------

        CSEG AT T00     ; gesamte Zeitsteuerung

        ajmp Timer0     ; zur Serviceroutine

 ; ---------------------

        CSEG AT X1      ; Power off Interrupt

        reti            ; nichts zum Testen
```

```
; ---------------------
       CSEG AT TO1     ; Timer 1 overflow
       reti            ; unused
; ---------------------
       CSEG AT SIO0    ; serielle Schnittstelle
       ajmp Ser        ; zur seriellen Routine
; ---------------------
       CSEG AT SIO1    ; I2C
       reti            ; unused
; ---------------------
       CSEG AT CT0     ; T2 capture 0
       reti            ; unused
; ---------------------
       CSEG AT CT1     ; T2 capture 1
       reti            ; unused
; ---------------------
       CSEG AT CT2     ; T2 capture 2
       reti            ; unused
; ---------------------
       CSEG AT CT3     ; T2 capture 3
       reti            ; unused
; ---------------------
       CSEG AT ADC     ; ADC fertig
       reti            ; unused - ADC wird ohne Interrupt verarbeitet
; ---------------------
       CSEG AT CM0     ; T2 compare 0
       reti            ; unused
; ---------------------
       CSEG AT CM1     ; T2 compare 1
       reti            ; unused
; ---------------------
       CSEG AT CM2     ; T2 compare 2
       reti            ; unused
; ---------------------
       CSEG AT TO2     ; T2 overflow
       reti            ; unused
; Interruptserviceroutinen
; ========================
Timer0: mov TL0,#207           ; Lowbyte reloaden
        mov TH0,#213           ; Highbyte reloaden
                               ; <--- alle 0.0117s = 1 tick
```

```
        djnz ftime,Tim_A        ; springe wenn nicht Blinken

                                ; <--- alle flash ticks

        mov ftime,#flash        ; ftime reloaden

        mov A,Latch0            ; LED-Einstellungen holen
        cpl A                   ; Komplement
        mov Latch0,A            ; Zurückspeichern
        orl A,DIP               ; auf Latch0 zusätzlich DIP-Anzeige
        acall mL0               ; und anzeigen
        mov A,Latch1            ; LED-Einstellungen holen
        cpl A                   ; Komplement
        mov Latch1,A            ; Zurückspeichern
        mov A,P4                ; Eingänge holen
        anl A,#00111111b        ; Ausgänge ausblenden
        orl A,Latch1            ; LEDs puls Eingänge
        acall mL1               ; Anzeige sichtbar machen
        mov A,P5                ; P5 holen
        cpl A1                  ; Komplement
        mov C,ACC.5             ; HMin kopieren
        orl C,A1                ; Oder
        mov A1,C                ; Zurück
        cpl A2                  ; Komplement
        mov C,ACC.4             ; HM kopieren
        orl C,A2                ; Oder
        mov A2,C                ; Zurück
        cpl A3                  ; Komplement
        mov C,Al                ; Alarm kopieren
        orl C,A3                ; Oder
        mov A3,C                ; Zurück

Tim_A:  reti                    ; auf ein Neues

Int00:  reti

Ser:    jbc TI,Ser_A            ; Transmit / springe und lösche
        jbc RI,Ser_B            ; Receive / springe und lösche
        cpl A3
        reti                    ; zurück, keine Ahnung was es wirklch war

Ser_A:  reti                    ; zurück / keine Transmitaktion

Ser_B:  mov R1,SOBuf            ; Echo senden
        mov SOBuf,R1
        reti

 ; Inititialisierung
 ; =================

Init:   mov SP,#Stack           ; Stack setzen
        mov P1,#0FFh            ; P1 = Eingang
        mov P3,#11011111b       ; P3 = Eingang, Ausgang
        mov P4,#00111111b       ; P4 = Eingang, Ausgang
        mov Latch0,#0FFh        ; alle LEDs aus (Bestimmung später)
        acall mL0               ; Latch 0 aktualisieren
        mov Latch1,#11110001b   ; alle LEDs und Ausgänge aus (Bestimmung später)
        acall mL1               ; Latch 1 aktualisieren
        mov ftime,#flash        ; Blinkzeit initialisieren
        mov ADCON,#00000110b    ; ADC-Grundeinstellung
        mov TMOD,#00100001b     ; Timer (0 & 1)-Grundeinstellung | ÄNDERUNG
        mov PCON,#0             ; kleinere Baudrate     | ÄNDERUNG
        mov SOCON,#01010000b    ; Serielle-Grundeinstellung | ÄNDERUNG
        mov TH1,#0FDh           ; Reloadvalue für 9600 bzw. 19200 baud
        mov IP0,#00000110b      ; Interruptprioritäten
        mov IP1,#0              ; Timer 2 alle auf 0
        mov IEN1,#0             ; kein Timer2 Interrupt
        mov TCON,#01010101b     ; Timer 0 starten und externe Interrupts edge-triggered

        mov ftime,#flash        ; ftime loaden
        mov out,#0              ; out loaden
        mov TL0,#207            ; Lowbyte loaden
        mov TH0,#213            ; Highbyte loaden
        mov Latch0,#01010101b   ; Muster speichern
        mov Latch1,#01010101b   ; Muster speichern

        mov IEN0,#10010011b     ; Interrupts erlauben

Go_A:   sjmp Go_A               ; nur die Interrupts "tun" was
```

```
 ; Unterroutinen
 ; =============

 ; mL1                          ; void mL1 (void)
 ; ***                          ; schreibt den aktuellen Wert von Latch1 ins Latch 1

mL1:    mov DPTR,#XL1           ; Adresse (#) des Latches 1 laden
        movx @DPTR,A            ; ins Latch 1 speichern
        ret

 ; mL0                          ; void mL0 (void)
 ; ***                          ; schreibt den aktuellen Wert von Latch0 ins Latch 0

mL0:    mov DPTR,#XL0           ; Adresse des Latches 0 laden
        movx @DPTR,A            ; ins Latch 0 speichern
        ret

END
```

Die Definitionen sind nahezu identisch mit denen des Hauptprogramms und werden daher hier nicht mehr erläutert[1]; gleiches gilt für die Tabelle der Vektoren[2], die Unterroutinen[3] und die Initialisierung[4]. Die Funktionen des Programms sind folgende:

- Alle Ausgänge (Latch0, Latch1, A1. . . A3) werden alle `ftime` ein bzw. ausgeschaltet. Dies wird durch Einstecken eines Sensors oder Aktivieren eines Sensors unterbrochen.
- Die serielle Schnittstelle empfängt Datenbytes und sendet sie unverändert zurück (Echo–Funktion).

Damit können alle Funktionsteile bis auf die Spannungsüberwachung, den ADC und den IR–Empfang getestet werden. Für den Test dieser Funktionen habe ich weitere Varianten dieses Programms geschrieben, in denen anstatt der Sensoren eben das Ziel über die LEDs angezeigt wird. Da diese Variationen aber keine besonderen Programmdetails enthalten, verzichte ich hier auf eine Auflistung.

13.2.2 Liste der Fehler und deren Behebung

Alle Tests der Hardware und der Software wurden mit Hilfe eines *AMIGA*–Computers durchgeführt. Zusätzlich stand der von mir gebaute Epromsimulator und die von mir programmierte zugehörige Software[5] zur Verfügung. Daher konnte die Entwicklung stark beschleunigt werden, weil das andauernde EPROM–Löschen und –Brennen entfallen konnte.

[1] siehe Kapitel 9.7.1.1 bis 9.7.1.10 ab Seite 97

[2] siehe Kapitel 9.7.2.1 auf Seite 105

[3] siehe Kapitel 9.7.2.5.2 und 9.7.2.5.3 auf Seite 116

[4] siehe Kapitel 9.7.2.3 auf Seite 114

[5] Sowohl der Epromsimulator als auch die zugehörige Software ist **nicht** Teil dieser Diplomarbeit. Daher werden weder die Software noch die Hardware hier näher erläutert. Weiters bleiben auch **alle** Rechte dieses Projekts uneingeschränkt bei C. Beck.

In der folgenden Auflistung sind alle gefundenen Hardwarefehler und deren Behebung aufgeführt. Zum Teil wurden Hardwarefehler jedoch erst während der Softwaretests gefunden; wegen der Übersichtlichkeit sind diese jedoch auch hier angeführt. Alle Fehler wurden dem handschriftlich geführten Entwicklungsprotokoll entnommen, das wegen einer medizinischen Überprüfung geführt werden mußte.

In der Auflistung wird die Fehlerbeschreibung durch ☹ und die Behebung durch ✓ gekennzeichnet.

- ☹ Messung von ca. 5V am GND–Anschluß des Batteriehalters.
  ✓ Verwechslung der Batterieanschlüsse behoben.
- ☹ Fehlen der LEDs für die Mousetastenkontrolle.
  ✓ LEDs und Vorwiderstände am Print hinzugefügt.
- ☹ Keine Funktion der Relaisausgänge.
  ✓ Hinzufügen von Pullup–Widerständen an allen drei Ausgängen. Weiters kam es durch das CAD–Programm zum „Mergen“ von Teilen des Adressbusses und des Ausgangsbusses. Daher mußten die entsprechenden Leitungen getrennt werden.
- ☹ LEDs des Latchs 0 reagieren nicht, die des Latches 1 falsch.
  ✓ In der Software wurde bei der Definition der externen Adressen eine Null vergessen, sodaß an Stelle von A15 A14 und an Stelle von A12 A13 angesprochen wurde. $\Longrightarrow$ Softwareverbesserung
- ☹ Serielle Schnittstelle sendet und empfängt nicht.
  ✓ Nach Messungen im Ruhezustand konnte ich herausfinden, daß die Übertragung der TXD– und RXD–Leitungen per Definition invertiert durchgeführt wird und die Ausgänge des Prozessors nichtinvertierend sind. Daher werden außer dem MAX232 keine weiteren Inverter in den Leitungen benötigt. $\Longrightarrow$ Inverter aus den Leitungen durch Überbrückungen entfernen.
- ☹ Keine Funktion des IR–Eingangs.
  ✓ Nach der Überprüfung der Pegel am IC SL486 stellt sich heraus, daß am Pin 7 auf die Versorgungsspannung vergessen wurde.
- ☹ Batterieverbrauch viel zu hoch
  ✓ Neben Softwaremaßnahmen (siehe Software–History im Kapitel 14 muß ein Widerstand in die Verbindung zwischen Batterie und Meßport 5 eingefügt werden, der den Strom im Power–Down–Modus einschränkt. Der Verbrauch konnte so im Testaufbau im Power–Down–Modus auf 29μA gesenkt werden.

13.3 Gehäuseherstellung und Einbau

Obwohl dieser erste Testprint nie für den Einbau in ein Gehäuse gedacht war, sollte der Print für Vorführzwecke in ein Gehäuse eingebaut werden. Daher mußten alle LEDs auf die Frontplatte verlegt werden. Da das Auslöten von 19 LEDs und einer IR–Diode eine sehr zeitaufwendige als auch für den Print gefährliche Arbeit gewesen wäre, wurden nur die Anschlüsse von den Dioden getrennt; diese verblieben aber am Print. Die Verbindung zwischen Frontplatte und Print wurde durch *sehr* viele Einzeldrähte verwirklicht.

Ein weiteres Problem stellte sich durch die über ein Eck angebrachten Klinkenbuchsen. Da die Buchsen durch die Seitenwände hindurchschauen müssen um ein Anstecken zu ermöglichen, mußten die Bohrungen erheblich größer (⌀10mm statt ⌀6mm) gemacht werden.

Zuletzt wurde die SUB–D–Buchse, die sich auf der gegenüberliegenden Seite der Klinkenbuchsen befindet, mittels eines „Verlängerungskables“ (Kabel mit 2 Sub–D–Steckern) an die Außenseite gelegt.

Da dieser Vorgang nicht zum Wiederholen gedacht ist (und auch wegen des hohen Zeitaufwandes nicht zu empfehlen ist) werden hier keine Bohr– oder Montagepläne angegeben. In einer zukünftigen Version des Prints wird auf einen Einbau des Prints in Gehäuse selbverständlich Rücksicht genommen, sodaß ein einfacher Einbau gewährleistet ist.

Zur einfacheren Vorstellung des Aussehens ist in Abbildung 13.1 die Beschriftung der Frontplatte zu sehen. Diese wurde auf eine Overheadfolie spiegelverkehrt aufgedruckt

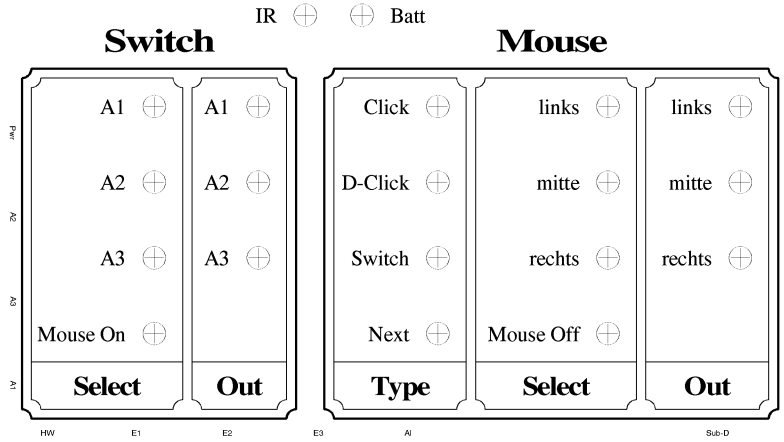

Abbildung 13.1: Beschriftung der Frontplatte im Maßstab 1:2. Die Positionen der LEDs bzw. der IR–Diode sind durch die Kreise mit den Kreuzen in der Mitte gekennzeichnet.

(wegen der Kratzfestigkeit) und wird durch die Befestigungsschrauben der Frontplatte

und die Fassungen der LEDs gehalten.

Die beiden Modis „Switch“ und „Mouse“ sind auch optisch durch die Rahmen getrennt. Am linken bzw. unteren Rand ist die Buchsenanordnung angegeben.

Kapitel 14

History des Programms „TheProg“

Im folgenden wird die Versionsgeschichte des Programms „TheProg“ beschrieben. Ab der Version 0.74 wurde das Programm mittels des Simulators auf der Hardware getestet; dieser Vorgang ist wieder im handschriftlichen Entwicklungsprotokoll festgehalten, woraus diese Aufstellung zum Teil entnommen ist.

0.70

- erste vollständige Version

0.71

- A1, A2, A3 von negierten Ausgängen auf normale (wegen Hardwareanforderung) verändert.
- `MLT`, `MMT`, `MRT` von negierten Ausgängen auf normale (wegen Hardwareanforderung) verändert.

0.72

- Völlige Neuprogrammierung des Mod5, da die von meinem Betreuer gelieferten Angaben über die EOG-Brille nicht stimmten.
- Einführung eines Mod8 = Mod5 + Scanning.
- Mehrere Verwechslungen von `MOn` und `MMon` behoben.

0.73

- Fehler durch vergessene Null bei der Adresse der externen Latches behoben.
- Fehler in der Initialisierung durch vergessene Definitionen der Ein– und Ausgänge der Ports behoben.

- Codeoptimierung durch Ersatz von `jb` und `clr` durch `jbc`.

0.74

- FatalError–LED–Einstellungen korrigiert.
- Initialisierung von `PSW` hinzugefügt.
- Initialisierung des Timers 0 `TH0`, `TL0` hinzugefügt.
- Einführung eines Bits `abusy`, damit während des Modustests durch die Interrupts keine Veränderungen an den Messages durchgeführt werden.
- Korrektur der LED–Einstellungen beim Programmieren.
- Korrektur der LED–Einstellungen bei der Initialisierung.
- Fehler bei der ADC–Auswertung (Warten auf Ready–Bit) behoben.
- Größer/Kleiner–Verwechslung beim Batterietest behoben.
- Abfrage–Bitmuster bei den Eingangsbuchsen korrigiert.
- Löschen der Message in `att_tst` hinzugefügt.
- Korrektur: bei aktivem Gerät 3 wurde beim Weiterschalten auf die Sel–LED vergessen.

0.75

- DIP–Schalterfunktionen invertiert, damit die Beschriftung des Schalters mit der Funktion übereinstimmt.
- Behandlung des MouseOn Menüpunktes im Switchmode hinzugefügt.
- Fehler bei der MouseOn–Menüpunkt–Erkennung (HMin $\leftrightarrow \overline{\text{HMin}}$) behoben.
- Entprellung von E3 hinzugefügt.

0.76

- Coldstart wird nun auch dann durchgeführt, wenn im RAM eine OK–Kennung nicht gefunden wird.
- Korrektur der Mousetastenansteuerung in `Clickit`, da auf eine Aktualisierung des Latches vergessen wurde.
- Fehler bei Menüpunkterkennung (nicht nach DIP–Einstellungen gefragt) behoben.
- Initialisierung von `stime` hinzugefügt.

- Fehler in Timerinterruptroutine im Prog–Mode behoben.
- Fehler bei LED–Initialisierung im Prog–Mode behoben.
- Fehler bei Interrupteinschränkung im Prog–Mode behoben.
- Fehlende Fehlermeldung im Prog–Mode hinzugefügt.
- Möglicher Konflikt beim Zugriff auf ADC zwischen Timer0–Interrupt und Prog behoben.
- Korrektur: Programm–Kennung beim Verlassen des Prog–Modes nicht gelöscht.
- Konflikt von Anschlußfehlerkennung und Kontrollscannen im Prog–Mode behoben.
- Völlige Neuprogrammierung der t_{ref}–Einstellung.
- A1 zum Test von t_{out} bei der Programmierung mitgeschaltet.
- Batterietest während der Programmierung völlig gestoppt.
- Umschaltung auf Mousemode im Mod2 verbessert (manchmal konnte ein FatalError ausgelöst werden).
- Initialisierung von `TelOn, SPC, STel` hinzugefügt.

0.77

- ABatt über die serielle Schnittstelle nicht mehr schreibfähig.
- Fehler im Readmodus behoben (Verwechslung von `jc` $\leftrightarrow$ `jz`).

0.78

- Antwort nach dem Reset–Code (serielle Schnittstelle) auch positiv (und nicht nur negativ).
- Völlige Neuprogrammierung der Resetauslösung über die serielle Schnittstelle, da es zu Hardwarekonflikten kam.
- Verwechslung von `PrgOn` und `PrOn` beim Batterietest behoben.
- Beim seriellen Read und Write müssen die Spezialregister (P1, P3, P4, P5) anders angesprochen werden, da diese nicht indirekt adressiert werden können.
- Fehler in `att_tst` bei der Messagebehandlung durch die neuen Messages der seriellen Schnittstelle behoben.
- Serielle Codes während des Programmierens ignorieren.

- Fehler beim IR–Empfang behoben; Variable als Register bei falsch aktivierter Bank angesprochen.
- Überwachung von `ScOn` hinzugefügt, damit Mod5 und Mod8 korrekt gesetzt werden können.

0.79

- Initialisierung von P0 und P2 wegen des Power–Down-Moduses hinzugefügt.

0.80

- Vor dem Power–Down alle Ports auf 0 gesetzt, da sonst ein zu hoher Verbrauch zu bemerken ist.

0.81 = 1.00

- Ausschalten der Mousetasten beim Verlassen des Mousemodes.

Teil IV

Beschreibungen des Gerätes

Kapitel 15

Funktionsbeschreibung des Multiswitches

In diesem Kapitel wird in kurzer Form der Funktionsumfang dieses Gerätes beschrieben. Diese Beschreibung unterscheidet sich vom Pflichtenheft[1] darin, daß die Beschreibung nicht technisch präzise und für den Normalanwender oder Servicetechniker gedacht ist.

15.1 Versorgung

Die Versorgung wird durch ein externes Gerät vorgenommen, das an die Versorgungsbuchse eine Spannung von mindestens 8V Gleichspannung oder 12V Wechselspannung bei einem höchsten Verbrauch von 0.2A liefert; die Spannung sollte 20V nicht übersteigen. Der Multiswitch wird durch eine Spannung bis 40V zwar nicht beschädigt, durch die hohe Wärmeentwicklung im Gerät kann es jedoch zu Störungen kommen.

15.2 Ausgabemodis

Das Gerät unterscheidet zwei Ausgabemodis: Den *Mousemode* und den *Switchmode*, wobei der Mousemode per Schalter völlig deaktiviert werden kann. Der Anwender kann immer nur einen der beiden Moden zur gleichen Zeit nutzen; der aktuelle Modus kann am Display dadurch erkannt werden, daß zumindest eine Selelektor–LED im entsprechenden Feld leuchtet. Der Wechsel zwischen den beiden Moden kann entweder durch einen eigenen Sensor oder mittels der Selektor–LEDs erfolgen.

[1] siehe Kapitel 4 ab Seite 19

15.2.1 Switchmode

Das Gerät befindet sich nach dem Einschalten immer in diesem Modus. Mit Hilfe des oder der Eingabesensoren kann die Selektor–LED zwischen den drei Ausgabegeräten und eventuell der Mousemodeaktivierung bewegt werden. Die auszulösende Aktion bezieht sich immer auf den Ausgang, der durch die Selektor–LED angezeigt wird. Die Ausgänge 1 und 2 können bei Betätigung der Aktion einen konstanten oder variablen Impuls — je nach Eingabemodus — abgeben, der Ausgang 3 wechselt bei der Aktion seinen Zustand.

Da nicht jeder Anwender drei Geräte anschließen will, können die Geräte 2 und 3 einzeln (in jeder Reihenfolge) deaktiviert werden, sodaß diese auch mit den Eingabesensoren nicht mehr ausgewählt werden können, was zu einer einfacheren Bedienung führt.

Sollte kein eigener Sensor für den Wechsel der beiden Ausgabemoden eingesteckt sein, so kann man den Mousemode mit Hilfe des virtuellen Gerätes „Mouse On“ aktivieren.

15.2.2 Mousemode

Mit Hilfe des oder der Eingabesensoren kann die Selektor–LED zwischen den vier Bedienmodis bewegt werden. Durch Auslösung einer Aktion wird die aktuelle, durch die Selektor–LED angezeigte, Mousetaste dem aktiven Bedienmodus entsprechend bedient. Es stehen die folgenden vier Bedienmodis zur Verfügung: Click, DClick (Doppel–Click), Switch (Ausgang wird ein– oder ausgeschaltet) und Next (nächste Mousetaste wird gewählt).

Da die mittlere Mousetaste bei Programmen selten gebraucht wird, kann diese deaktiviert werden, was wiederum zu einer vereinfachten Bedienung führt.

Sollte kein eigener Sensor für den Wechsel der beiden Ausgabemoden eingesteckt sein, so kann man den Switchmode mit Hilfe der virtuellen Mousetaste „Mouse Off“ aktivieren. Beim Verlassen des Mousemodes werden auf jeden Fall alle Mousetasten deaktiviert.

15.3 Eingabemodis

Um dieses Gerät mit nahezu allen bekannten Sensoren bedienen zu können, wurden acht verscheidene Varianten der Sensorbedienung implementiert.

Modus 1: Nur ein einziger Sensor wird in die Buchse E1 eingesteckt.

Einen Wechsel des aktiven Ausgangs kann durch längeres Halten des Sensors erreicht werden. Solange man den Sensor aktiviert hat, springt der Ausgang immer um eins weiter. Eine Aktion löst man durch kurzes Drücken aus.

Modus 2: Je ein Sensor wird in die Buchsen E1 und E3 eingesteckt.

Pro Aktivierung des Sensors in E3 springt der Ausgang um eins weiter. Im Switchmo-

de werden die Ausgänge 1 und 2 solange aktiviert, wie der Sensor in E1 gehalten wird, im Mousemode wird pro Aktivierung des Sensors eine Aktion ausgelöst.

Modus 3: Nur ein Sensor wird in die Buchse E2 eingesteckt.
Den nächsten Ausgang erreicht man durch zweimaliges „rasch“ aufeinanderfolgendes Betätigen des Sensors, eine Aktion löst man durch einmaliges Drücken des Sensors aus.

Modus 4: Je ein Sensor wird in die Buchse E2 und E3 eingesteckt.
Den nächsten Ausgang erreicht man wie im Modus 2 mit Hilfe des in der Buchse E3 eingesteckten Sensors. Eine Aktion wird mit Hilfe des Sensors in E2 durch zweimaliges „rasch“ aufeinanderfolgendes Betätigen ausgelöst; einfach Betätigungen werden ignoriert.

Modus 5: Kein Sensor wird in die Buchsen E1 bis E3 eingesteckt.
Zur Steuerung dient ein IR–aussendendes Gerät. Mit Hilfe eines Codes wird auf den nächsten Ausgang weitergeschaltet, mit Hilfe eines anderen eine Aktion ausgelöst.

Scanmodis: Alle bisherigen Eingabemoden können mit Scannen im Mousemode (nicht im Switchmode!) kombiniert werden. Durch Aktivierung des Scannens (mit einem DIP–Schalter im Gerät) wird das Weiterschalten auf den nächsten Bedienmodus automatisch erledigt. Die Auslösung einer Aktion kann weiterhin wie im Modus definiert erfolgen.

15.4 Programmierungen

Neben der Möglichkeit Ausgänge, die mittlere Mousetaste, den gesamten Mousemodus und den Scanmodus zu deaktiviern, können noch einige Zeiten eingestellt werden, um eine bessere Anpassung an den Anwender zu ermöglichen.

Der Programmiermodus wird durch einen DIP–Schalter im Gerät aktiviert. In diesem Modus ist eine normale Bedienung nicht mehr möglich und vier Zeiten können an den Anwender angepaßt werden.

15.5 Fehlermeldungen

Wenn in einem Bedienungsmodus ein Sensor nicht den Definitionen nach gedrückt wird (z.B. der Sensor im Modus 2 zu lange gehalten wird), oder eine Kombination von Sensoren in die Buchsen eingesteckt wird die keinen Modus ergeben, so werden durch Blinkzeichen

drei Fehlermeldungen ausgegeben. Sollte einmal ein schwerer interner Fehler auftreten, so schaltet sich das Gerät ab und kann nur durch eine Unterbrechung der Stromversorgung wieder reaktiviert werden.

15.6 Submodule

Der Multiswitch ist dafür ausgelegt weitere Submodule anzusprechen. Derzeit sind zwei Systeme geplant und werden von dieser Version des Multiswitches auch unterstützt: ein PC–Terminalprogramm und ein GSM–Handy–Ansteuergerät. Diese Geräte werden an die Sub–D–Buchse angeschlossen und beeinflussen zum Teil die Funktion des Multiswitches. Für eine eingehendere Beschreibung dieser Geräte und der Zusammenarbeit mit dem Multiswitch siehe den zugehörigen Beschreibungen.

Kapitel 16

Bedienungsanleitung

In diesem Kapitel wird eine kurze, sehr einfach gehaltene Bedienungsanleitung angegeben. Diese Version ist noch nicht für den Vertrieb geeignet, da keine optische Aufbereitung vorhanden ist und Beispiele fehlen. Dies ist zu diesem Zeitpunkt der Entwicklung noch nicht sinnvoll, da eine marktreife Version auch noch aussteht. Da durch die Weiterentwicklung des Gerätes zumindest die Bedienungsoberfläche verändert werden wird, sollte diese Kurzanleitung genügen.

Weiters ähnelt diese Anleitung zum Teil der Funktionsbeschreibung aus dem vorangegangenen Kapitel. Dies ist durchaus sinnvoll, da die Bedienungsanleitung unabhängig von dieser Diplomarbeit verwendet werden soll, sodaß Wiederholungen auftreten müssen.

16.1 Allgemeines

Dieses Gerät ist für den mobilen Betrieb konzipiert und soll dem Anwender die Bedienung des Computers mit Hilfe eines Mouseemulators erleichtern und die Möglichkeit mehrere Geräte mit Hilfe eines (oder zweier) Sensoren eröffnen.

16.2 Versorgung

Die Versorgung wird durch ein externes Gerät vorgenommen, das an die Versorgungsbuchse eine Spannung von mindestens 8V Gleichspannung oder 12V Wechselspannung bei einem höchsten Verbrauch von 0.2A liefert; die Spannung sollte 20V nicht übersteigen. Der Multiswitch wird durch eine Spannung bis 40V zwar nicht beschädigt, durch die hohe Wärmeentwicklung im Gerät kann es jedoch zu Störungen kommen.

16.3 Eingabemodis

Um dieses Gerät mit nahezu allen bekannten Sensoren bedienen zu können, wurden acht verschiedene Varianten der Sensorbedienung implementiert. Um nicht jeden Eingabemodus in beiden Arbeitsmodis erläutern zu müssen, werden die beiden Aktionen „Weiter“ (mit ▼ gekennzeichnet) und „Aktion“ (mit ● gekennzeichnet) definiert. Bei der Erläuterung der Arbeitsmodis kann daher für jeden Eingabemodus mit diesen beiden Aktionen die Bedienung erklärt werden.

Modus 1: Nur ein einziger Sensor wird in die Buchse E1 eingesteckt.

▼ Sensor in E1 länger gedrückt halten.
Solange der Sensor gedrückt bleibt, wird immer auf den nächsten Ausgang weitergeschaltet.

● Sensor in E1 kurz drücken.

Modus 2: Je ein Sensor wird in die Buchsen E1 und E3 eingesteckt.

▼ Sensor in E3 kurz drücken.

● Sensor in E1 drücken.
Im Switchmode werden die Ausgänge 1 und 2 so lange aktiviert, wie der Sensor gehalten wird.

Modus 3: Nur ein Sensor wird in die Buchse E2 eingesteckt.

▼ Sensor in E2 2× aufeinanderfolgend kurz drücken.

● Sensor in E2 1× drücken.

Modus 4: Je ein Sensor wird in die Buchse E2 und E3 eingesteckt.

▼ Sensor in E3 kurz drücken.

● Sensor in E2 2× aufeinanderfolgend kurz drücken.
Wenn der Sensor nur einmal betätigt wird, erfolgt keine Aktion.

Modus 5: Kein Sensor wird in die Buchsen E1 bis E3 eingesteckt.

▼ Code 1

● Code 2
Zur Steuerung ist zusätzlich ein geeignetes IR–aussendendes Gerät (z.B. EOG–Brille) notwendig.

Wenn im Mousemode das Scannen aktiviert ist, verändern sich die oben aufgelisteten Eingabemodis nur insofern, als daß die Funktion [▼] nicht mehr benötigt wird (und auch ignoriert wird), da [▼] vom Gerät selbst periodisch ausgeführt wird.

16.4 Arbeitsmodis

Das Gerät kann in den Modis Switch und Mouse betrieben werden, wobei immer nur einer der beiden Modis zur selben Zeit aktiv sein kann. Am Display sind die beiden Bedienfelder durch zwei Rahmen optisch getrennt. Der aktuelle Modus ist daran erkennbar, daß nur in diesem Feld eine LED in der Select–Reihe leuchtet. Der Wechsel zwischen den beiden Modis kann entweder mit einem Sensor in der Buchse HW extern erfolgen, oder mit einem virtuellen Gerät bzw. einer virtuellen Mousetaste.

16.4.1 Switchmode

Dieser Modus wird automatisch beim Einschalten des Gerätes aktiviert. Mit Hilfe von [▼] kann man den nächsten Ausgang selektieren; dies wird durch eine leuchtende LED neben dem entsprechenden Ausgang signalisiert. Da die Ausgänge 2 und 3 abschaltbar sind, kann es vorkommen, daß ein oder mehrere Ausgänge nicht selektierbar sind. Die vierte Selektionsdiode mit der Bezeichnung Mouse On kann nur selektiert werden, wenn kein Sensor in der HW–Buchse eingesteckt ist und der Mousemode nicht global deaktiviert ist.

Einen Ausgang kann man mit [●] aktivieren, im Falle des virtuellen Gerätes Mouse On wird bei einer Aktivierung in den Mousemode gewechselt. Der aktuelle Zustand der Ausgänge wird durch LEDs in der Reihe Out gekennzeichnet. Der Ausgang 3 stellt eine Ausnahme dar, da dieser nicht pro [●] einen Impuls abgibt, sondern seinen Zustand bei jedem [●] wechselt.

16.4.2 Mousemode

Dieser Modus basiert auf dem gleichen Bedienungssystem wie der Switchmode, nur daß ein Zweiebenensystem verwendet wird. Mit den LEDs in der Spalte Type wird die Art der Mousetastenaktivierung ausgewählt, wobei die Typen Click, DClick (Doppel–Click), Switch (Ein/Ausschalten) und Next zur Verfügung stehen. Um die nächste Aktivierungsart zu wählen betätigt man [▼].

Die zweite Ebene besteht wieder aus den vier Selektor–LEDs in der Reihe Select, wo die drei Mousetasten Links, Mitte, Rechts und die virtuelle Taste Mouse Off ausgewählt werden kann. Den nächsten Ausgang wählt man aus, indem man in der Reihe Type die Funktion

Next auswählt und mit [●] auslöst; mit jedem Drücken von [●] springt die Selektor–LED auf den nächsten Ausgang.

Möchte man den Modus Mouse verlassen, wenn kein eigener Sensor dafür in der HW–Buchse eingesteckt ist, so wählt man mit Hilfe von Next die virtuelle Mousetaste Mouse Off aus, drückt [▼] einmal um auf eine der Ausgangsaktionen zu kommen (z.B. Click) und löst diese mit [●] aus. Beim Verlassen des Moduses Mouse werden alle Mousetasten deaktiviert. Sollte einmal irrtümlich der Modus Switch gewählt worden sein, so kann man sofort wieder in den Modus Mouse gelangen, indem man [●] betätigt, da im Modus Switch immer noch das virtuelle Gerät Mouse On aktiv ist.

16.5 Fehlermeldungen

Es gibt drei Fehlermeldungen, die für den Anwender von Interesse sind. Diese werden mit Hilfe der LEDs in den Spalten Type und Select des Moduses Mouse dargestellt. Die

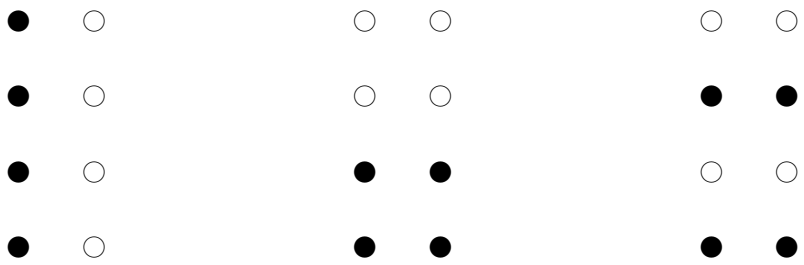

Abbildung 16.1: Darstellungen der Anzeige bei Fehlermeldungen: links: Anschlußfehler — mitte: Bedienungsfehler 1 — rechts Bedienungsfehler 2

Meldungen unterscheiden sich nur durch das Blinkmuster der angegebenen acht LEDs.

Anschlußfehler: Diese Fehlermeldung wird immer angezeigt, wenn in den Eingangsbuchsen E1 bis E3 Sensoren in einer Kombination eingesteckt sind, die nicht auf einen Eingabemodus führen (siehe Abbildung 16.1 links).

Bedienungsfehler 1: Diese Fehlermeldung wird angezeigt, wenn der Sensor in E2 beim ersten Drücken zu lange aktiviert ist (siehe Abbildung 16.1 mitte).

Bedienungsfehler 2: Diese Fehlermeldung wird angezeigt, wenn der Sensor in E2 beim zweiten Drücken zu lange aktiviert ist (siehe Abbildung 16.1 rechts).

Sollte einmal ein interner schwerer Fehler auftreten, so schaltet sich das Gerät ab und kann nur nach einer Stromunterbrechung wieder gestartet werden. Solche Fehler

sollten eigentlich nie auftreten, außer bei unsachgemäßer Handhabung mit einem PC–Steuerprogramm.

16.6 Erweiterungen

Der Multiswitch hat eine serielle Schnittstelle an der Sub–D–Buchse (die auch von den Mousetasten verwendet wird). Über ein spezielles Kabel können sowohl die Mousetasten als auch die externen Erweiterungen angeschlossen werden.

Derzeit sind zwei Erweiterungen für die Zusammenarbeit mit dem Multiswitch geeignet; durch ROM–Updates können jedoch auch zukünftige Erweiterungen verwendet werden.

PC–Schnittstelle: Der Multiswitch kann mit jedem handelsüblichen PC verbunden werden. Das mitgelieferte Programm kann die Einstellungen des Multiswitches speichern und verändern; weiters kann auch die Batteriespannung angezeigt werden.
Diese Erweiterung ist nur für Servicetechniker gedacht.

GSM–Handy–Schnittstelle: Mit diesem zusätzlichen Gerät kann auch ein GSM–Handy mit dem Multiswitch gesteuert werden.

Für weitergehendere Informationen wenden Sie sich bitte an den zuständigen Servicetechniker.

16.7 Einstellungen

Der Multiswitch ist an den Anwender anpaßbar. Dazu wurden die wichtigsten Zeiten entweder über die PC–Schnittstelle oder direkt am Gerät verändert. *Dieser Vorgang sollte nur von einem geschulten Servicetechniker durchgeführt werden!*

Um Einstellungen vornehmen zu können, muß man das Gehäuse öffnen und den neben dem Prozessor befindlichen 8fach DIP–Schalter bedienen; die Belegung des Schalters ist in der Tabelle 16.1 angegeben. Die mit ϟ gekennzeichneten DIP–Schalter können zu jeder Zeit betätigt werden und haben sofort Wirkung auf die Bedienung. Dabei ist jedoch immer darauf zu achten, daß diese Einstellungen bei der *nächsten* Aktion berücksichtigt werden. D.h. wenn z.B. gerade die mittlere Mousetaste ausgewählt ist und man schaltet diese mit Hilfe des DIP–Schalters 7 aus, so bleibt diese aktiviert; sie kann aber nicht mehr beim nächsten Versuch angewählt werden.

Die Einstellung der Karrenzzeit mit den beiden DIP–Schaltern 1 und 2 wird erst beim Einschalten des Programmiermoduses aktiv!

1 2	Karrenzzeit-Einstellung als Verhältnis der Referenzzeit	
3 ϟ	out 3 on/off	falls das 3. Gerät nicht angeschlossen ist
4 ϟ	out 2 on/off	falls das 2. Gerät nicht angeschlossen ist
5 ϟ	mouse on/off	Mousemode insgesamt ein/aus
6 ϟ	scan on/off	schaltet Scan-Mode im Mousemode ein/aus
7 ϟ	MouseMitte on/off	schaltet den mittleren Mousebutton ein/aus
8	progable on/off	schaltet die Programmiermöglichkeit ein/aus

Tabelle 16.1: Belegung der DIP–Schalter

16.7.1 Kurzerläuterung der charakteristischen Zeiten

Aus der Abbildung 16.2 können die Zeiten t_{ref}, t_{out} und t_K im Modus 1 beispielhaft ersehen werden.

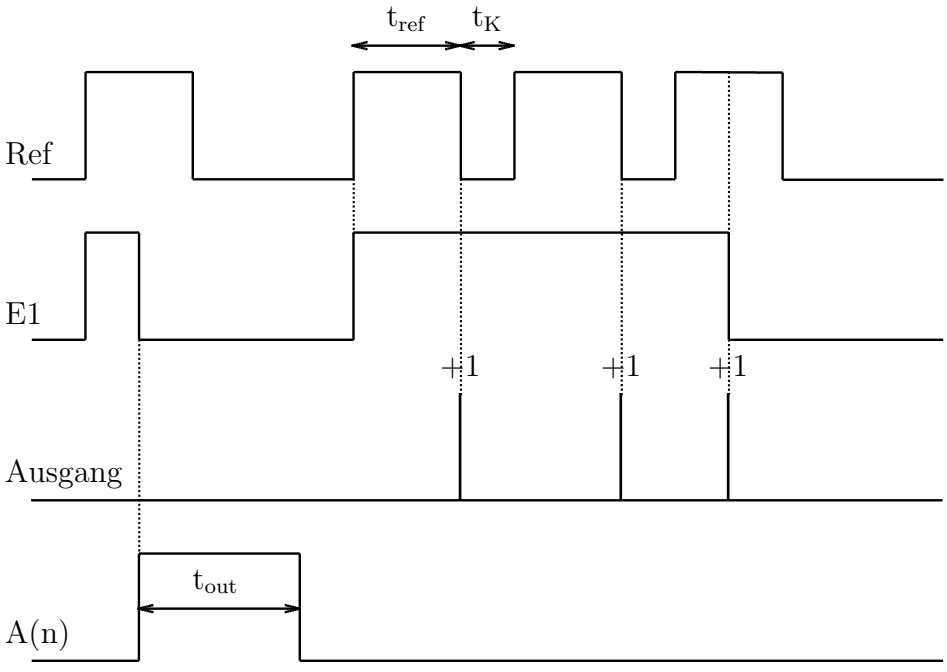

Abbildung 16.2: Darstellung zur Erläuterung der charakteristischen Zeiten

16.7.2 Programmiermodus

Der Programmiermodus kann durch den DIP–Schalter 8 aktiviert werden. Damit wird automatisch die aktuelle Einstellung der Karrenzzeit übernommen! Es kann immer nur eine Einstellung zur selben Zeit vorgenommen werden. Die Einstellungsauswahl wird durch Einstecken von Sensoren in die Buchsen vorgenommen, wenn kein Einstellungsmodus aktiv ist, wird die Anschlußfehlermeldung angezeigt.

Karrenzzeit: Kein Sensor eingesteckt, Anschlußfehlermeldung aktiv

Die Karrenzzeit t_K wird mittels der DIP–Schalter 1 und 2 eingestellt, wobei die Position 2 das höherwertige Bit darstellt. Die Karrenzzeit ergibt sich aus der Formel 16.1.

$$t_K = \frac{t_{ref}}{2^{Biteinstellung}} \tag{16.1}$$

Scanzeit: ein Sensor in der Buchse HW

Mit Hilfe des Potentiometers am Print kann die Scanzeit eingestellt werden. Zur Kontrolle der Zeit wurde ein Lauflicht in den Spalten Type und Select im Modus Mouse eingerichtet.

Achtung: Die Einstellung des Potentiometers muß nicht identisch mit dem aktuellen Wert vor der Programmierung sein. Daher kann sich alleine durch die Aktivierung der Scanzeiteinstellung eine starke Änderung der Einstellung ergeben!

Referenzzeit: ein Sensor in der Buchse E1

Die Referenzzeit t_{ref} wird mittels eines Sensors in der E1–Buchse eingestellt. Die Zeit ist genauso lange wie der Sensor gehalten wird und wird bei jedem Versuch neu definiert. Die Zeit kann maximal 3 Sekunden betragen. Sollte dieses Limit überschritten werden, so ist eine Fehlermeldung (alle LEDs der Reihen Type und Select im Modus Mouse eingeschaltet) zu sehen; ein fehlgeschlagener Versuch verändert die Einstellung nicht.

Ausgangszeit: ein Sensor in der Buchse E2

Die Ausgangszeit t_{out} wird mittels eines Sensors in der E2–Buchse nach dem gleichen Schema wie die Referenzzeit eingestellt.

Weitere Zeiten können mit dem PC über die serielle Schnittstelle verändert werden (siehe Beschreibung des zugehörigen Programms).

16.8 Reset

Alle Zeiten bleiben auch bei einer Unterbrechung der Spannungsversorgung erhalten. Einen Generalreset (alle Einstellungen werden gelöscht) kann man wie folgt durchführen: Beim Anstecken der Stromversorgung dürfen nur je ein Sensor in den Buchsen E1 und E2 eingesteckt und gehalten werden.

16.9 Batteriewechsel

Die eingebaute Litiumbatterie sollte für ca. 2 Jahre die Einstellungen ohne externe Stromversorgung erhalten können. Wenn die Batterie leer wird, leuchtet am Display die Batt–LED. Sollte das der Fall sein, so ist die Batterie *vom Servicetechniker* wie folgt zu wechslen: Die Lithiumbatterie vom Type CR2450 wird bei angesteckter und ausreichender externer Versorgung aus dem Batteriehalter genommen und durch eine neue ersetzt. Sollte während des Tausches die Versorgung ausfallen, gehen alle Einstellungen verloren; auf richtige Polung beim Einsetzen ist zu achten!

Teil V

Anhang

Nachwort

Diese Diplomarbeit hat zwar länger gedauert als zuerst geplant (ca. 10 Monate) und konnte leider nicht alle Anforderungen erfüllen, hat jedoch ein für Schwerstbehinderte sehr hilfreiches Gerät hervorgebracht, daß durch die Offenheit der seriellen Schnittstelle auch die Möglichkeit für zukünftige Erweiterungen bietet. Für die Marktreife fehlen noch ausgiebige Tests in der Praxis, eine neue Generation des Prints (Aufbau in SMD) und ein ansprechendes Gehäuse[1].

Trotz der vielen Widrigkeiten denen man bei Verhandlungen mit Firmen ausgesetzt ist und dem Ärger über mangelnde Unterlagen zu Bauteilen und Programmen, bin ich froh, diese Diplomarbeit angenommen und ausgeführt zu haben, da sie Schwerstbehinderten ein Leben lang eine erhebliche Verbesserung ihrer Lebensqualität ermöglicht.

Im Laufe der Arbeit habe ich, neben wenigen negativen Ausnahmen, viele Leute kennengelernt, denen ich an dieser Stelle danken möchte:
Herr Ing. Prulamp war für mich als praktischer Betreuer immer da, Herr Dr. Thoma als Betreuer im AKH als auch Herr Prof. Wintner an der TU–Wien haben mir bei vielen bürokratischen Hindernissen geholfen, Herr Dipl. Ing. Uhlirz von der Firma Alcatel hat mir schnell wertvolle Informationen zum GSM07 Standard verschafft, Herr Timo Saraketo von der Firma Nokia in Finnland hat mir durch Informationen und Informationsmaterialien zu Nokia–Handys sehr geholfen, sowie die Firma Philips generell, die mich mit Programmen und Informationen zum Prozessor sehr unterstützt hat.
Zuletzt seien hier noch meine Familie und meine Freunde erwähnt, die mich während der ganzen Diplomarbeit unterstützt haben.

Als persönliche Empfehlung für alle anderen Studenten, die eine Diplomarbeit in diesem Umfang schreiben müssen/wollen, sei hier erwähnt, daß dieses Dokument mit LaTeX erstellt

[1] Anmerkung vom Jänner 1999: Seit meiner Diplomarbeit habe ich 1 Jahr an der Marktreife des Multiswitches gearbeitet, sodaß dieses Projekt mit den hier beschriebenen Funktionen abgeschlossen ist. Während der Weiterentwicklung sind weitere Ideen für zukünftige Verbesserungen entstanden, die jedoch noch nicht verwirklicht werden konnten. Weitere Informationen dazu sind auf meiner Homepage (http://www.fortec.tuwien.ac.at/chb zu finden.

wurde, was eine problemlose und einfache Möglichkeit zur Bearbeitung großer Dokumente mit Bildern, Formeln und Tabellen darstellt.

Literaturverzeichnis

[1] *Data Handbook IC20 – 1997*, Philips Components Division, 1996

[2] *Datenblatt des LM385*, National Semiconductors, Dezember 1994, zu beziehen unter http://www.national.com

[3] *Datenblätter des MV500, MV601 und SL486*, Plessey Semiconductors, zu beziehen unter http://www.plessey.com

[4] *final draft European Telecommunication Standard (ETS)*, Special Mobile Group (SMG) Technical Committee (TC) of the European Telecommunications Standards Institute (ETSI)

[5] *Informationsblätter*, Firma Gewa AB

[6] *Informationsblatt*, Firma Schweizerische Stiftung Elektronischer Hilfsittel für Behinderte

[7] *Integrated Circuits Databook*, Maxim, 1989; auch einzeln zu beziehen unter http://www.maxim.com

[8] *Katalog 1995/96*, Firma Incap GmbH.

[9] *PCB83C552/562 User Manual*, Philips Components Division, 1989

[10] *PIC167X Databook – 1996*, Arizona Mircochip, 1996

[11] *PCMCIA für Entwickler*, Michael T. Mori, 1995, ISBN 3–88229–067–6

Anhang A

80C562

In diesem Kapitel sind die Darstellungen des Prozessors 80C562 enthaten, die im Kapitel 6 ab Seite 47 wegen der Übersichtlichkeit nicht inkludiert werden konnten.

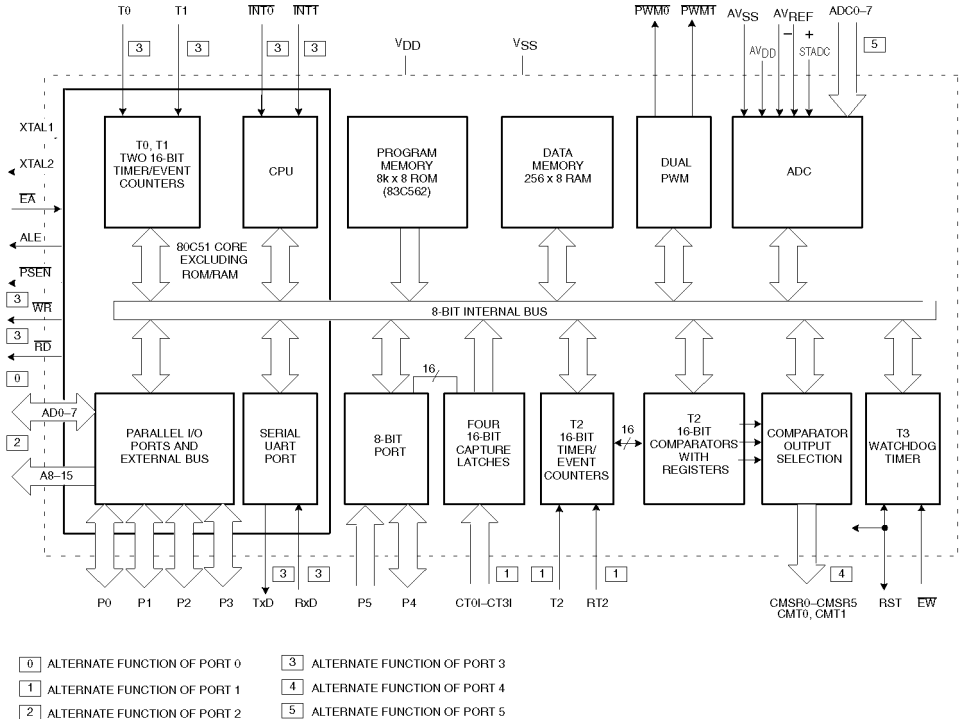

Abbildung A.1: Blockdiagramm des 80C562 [1]