Praktikum Kommunikationstechnik

Versuch: Digitale Vermittlungstechnik 2

1 Versuchsziele

Kennen lernen:

- eines Software-Moduls zur Schleifenauswertung
- eines Software-Moduls zur Wahlbewertung
- eines Software-Moduls zur Ruf- und Hörtonsteuerung
- eines Software-Moduls zur Zustandsverwaltung der Teilnehmer

Versuchsaufbau

Die Vermittlung ist in einem Kompaktgehäuse untergebracht und enthält:

- Netzteile zur Erzeugung der Betriebsspannungen (2,5 V, 3,3 V und 20 V),
- ein Taktmodul zur Erzeugung notwendiger PCM-Takte,
- einen LAN- und USB-Anschluss zur Kommunikation mit dem Steuerrechner,
- einen Mikrokontroller für die Koordination des Hardwarezugriffs,
- vier a/b-Teilnehmersätze,
- vier RJ45-Anschlüsse um die Endgeräte an der Vermittlung anzuschließen.

Inhaltsverzeichnis

Inhaltsverzeichnis i
Abbildungsverzeichnis iii
1 Einleitung 1
1.1 Motivation 1
2 System Endgerät/Vermittlung 3
2.1 Zu Abbildung 14a: 3
2.2 Zu Abbildung 14b: 3
2.3 Zu Abbildung 14c: 4
3 Funktionalität des Moduls SIG 7
3.1 Hardware 8
3.2 Software 8
4 Programmodul SIG 9
4.1 Zustands- und Wahlerkennung 9
4.1.1 Grundlagen 9
4.1.2 Festlegung der Bearbeitungsperioden 10
4.1.3 Automatenstruktur 11
4.1.4 Softwarerealisierung 16
4.2 Hörton- und Rufsteuerung 19
4.2.1 Grundlagen 19
4.2.2 Realisierung der Hörton- und Rufsteuerung 21
5 Versuchsdurchführung 25
5.1 Allgemeines 25
5.2 Die Kommunikation zwischen Modulen 25
5.3 Der Prozessmanager 26
5.4 Die Call Control 28
Anlagen 31
Literaturverzeichnis 75

Abbildungsverzeichnis

Abbildung 1: Anschlussarten für PABX und OVST 1
Abbildung 2: Hard-Software-Struktur der a/b-Signalgabe 7
Abbildung 3: Meldungsinterfaces SIG-CC, CC-MAR 8
Abbildung 4: Zusammenhang zwischen logischen Schleifenzuständen und vermittlungstechnischen Ereignissen..... 9
Abbildung 5: Toleranzschema für Nummernschalterwahl 10
Abbildung 6: Prinzip der Abtastmethode.....11
Abbildung 7: Automatenstruktur zur Zustands- und Wahlerkennung 12
Abbildung 8: Aufgabenstellung für Automat1.....13
Abbildung 9: Graph und Tabelle des Automaten 13
Abbildung 10: Graph des Automaten zur Zustandserkennung und Wahlbewertung 15
Abbildung 11: Taktinterpretation für Hard- und Softwareautomaten 16
Abbildung 12: Zeitbedingungen für das Besetzt- und Freizeichen nach Q.35 20
Abbildung 13: Steuersequenzen für ausgewählte Kennzeichen 21
Abbildung 14: System Endgerät/Vermittlung 31

2 Einleitung

2.1 Motivation

Die Leistungen eines Netzes nehmen die Teilnehmer vermittels ihrer Endgeräte in Anspruch. Vor der eigentlichen Nutzung müssen Informationen zur Steuerung der Netzeinrichtungen (Vermittlungen) zwischen Endsystemen und dem Netz (Nutzer-Netz-Schnittstelle) ausgetauscht werden.

Eine weit verbreitete Nutzer-Netz-Schnittstelle zu privaten Vermittlungen (Nebenstellenanlagen) bzw. zu Vermittlungen des öffentlichen Netzes ist der a/b-Anschluss. In Abbildung 1 sind wesentliche Nutzer-Netz-Schnittstellen im Bereich von Nebenstellenanalagen und Ortsvermittlungsstellen dargestellt.

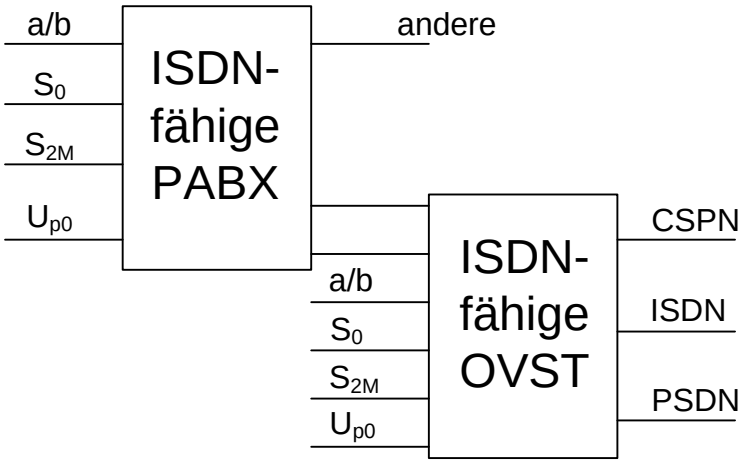


Abbildung 1: Anschlussarten für PABX und OVST

In diesem Arbeitsblatt soll gezeigt werden, wie die Signalgabe für a/b-Anschlüsse in digitalen Vermittlungen realisiert wird.

3 System Endgerät/Vermittlung

In Abbildung 14 sind für die a/b-Signalgabe dargestellt:

- das Prinzip (Abbildung 14a),
- eine einfache Realisierung (Abbildung 14b),
- die Elemente bzw. Schnittstellen dieses Kennzeichensystems (Abbildung 14c).

Aus dieser Abbildung geht auch die 3-Teilung in Endeinrichtung, Übertragungseinrichtung und Netzeinrichtung hervor.

3.1 Zu Abbildung 14a:

Die Hardware jeder Vermittlung kann man in die funktionsteile:

- Fernmeldetechnische Peripherie (FMTP),
- Vermittlungstechnische Peripherie (VMTP),
- Koppelnetz (KN) und
- Steuerung (ST) einteilen.

Die FMTP enthält die verschiedensten Leitungssätze. Aufgabe dieser Sätze ist die Anpassung an die Übertragungseinrichtung (Speisung, Kennzeichenerkennung/ -aussendung usw.) einerseits und andererseits die Anpassung an das Koppelnetz und die Steuerung.
Die Steuerung unabhängig von der hardwaremäßigen Realisierung wird durch drei Programmsysteme:

- Signalgabe (SIG),
- Belegungsbearbeitung (BBA) / Call Control (CC) und
- Markierung (MAR) bezüglich der Hauptfunktion realisiert.

Hier soll für die a/b-Signalgabe hauptsächlich die Satzhardware als Teil der FMTP und das Programm SIG als Teil der Steuerungssoftware betrachtet werden.

3.2 Zu Abbildung 14b:

Dieses Teilbild zeigt die Signalgabeeinrichtungen eines Fernsprechendgerätes sowie eine einfache, aber funktionierende Realisierung eines a/b-Teilnehmersatzes. Die Funktionen des Teilnehmersatzes werden üblicherweise durch das Kunstwort BORSCHT benannt.
BORSCHT-Funktionen sind: Battery, Overload Protection, Ringing, Signalling, Codec/Filter, Hybrid, Test.

Teilnehmerspeisung:

Von Zentralbatterie (ZB), über $OK_{1,2}$; $L_{1,2}$; $R_{SP1,2}$; a/b-Adern. Die im Speisezweig liegenden Bauelemente haben die folgenden Funktionen:

- $L_{1,2}$:NF-Abblockung der Zentralbatterie ZB,
- $R_{SP1,2}$:Speisestrombegrenzung,
- $OK_{1,2}$:Schleifenzustandsdetektoren (physikalisch-logischer Konverter).

Rufspeisung:

Die Speisequelle ist ein 25 Hz-Generator, der über einen Trafo symmetrisch zwischen Erde und b-Ader über das Relais R angeschaltet wird.

Die Strompfade sind:

- positive Halbwelle: G, ZB. D_3, a-Ader, Wecker, b-Ader, r_1, G,
- negative Halbwelle: G, r_1, b-Ader, Wecker, a-Ader, D_1, G.

Der Ruf wird über einen Relaistreiber aufgeschaltet, wenn die Variable r die Belegung HIGH hat. Die Rufsequenz wird softwaremäßig realisiert.

Schleifenzustandserkennung (physikalisch-logische Konvertierung)

Diese Logik dient dazu, im Ruf- und Nichtrufzustand einen Gleichstromfluss in den a/b-Adern zu detektieren und als logische Signale (LOW oder HIGH) am Satzinterface zur Verfügung zu stellen.

- Rufzustand :Optokoppler1 s_a = | 1 Schleife
  | 0 $\overline{\text{Schleife}}$
- Nichtrufzustand: :Optokoppler1,2 $s_{a,sb}$ = |1 1 Schleife, ET
  |1 0 Schleife, $\overline{\text{ET}}$
  |0 1 ungültig
  |0 0 $\overline{\text{Schleife}}$, $\overline{\text{ET}}$

Durch Verwendung der Optokoppler erreicht man gleichzeitig eine galvanische Trennung zwischen Steuerung und Satz.

3.3 Zu Abbildung 14c:

In diesem Teilbild sind die Elemente und Schnittstellen des Signalgabesystems a/b dargestellt. Im Folgenden wird kurz auf die einzelnen Funktionsblöcke eingegangen.

Konvertierungsfunktion (physikalisch-logische):

- Indikation und Bewertung der Gleichstromschleifensignale,

- Konvertierung richtig erkannter elektrischer Signale (Stromfluss in Schleifen) in logische (s_a,s_b) und deren Übergabe an die SIG-Funktion,
- Annahme logischer Signale von der SIG-Funktion (hier r) und Ausgabe über das physikalische Signalinterface.

Signalisierungsfunktion (SIG):

Die Signalisierungsfunktion besteht aus einer rechnertypischen Hardware (z.B. PIO) zur Abfrage der Schleifenzustände (s_a,s_b) bzw. zur Rufsteuerung über die Variable r und einer speziellen Software mit folgend Aufgaben:

- Verarbeitung der logischen Signale s_a,s_b bezüglich Dauer sowie Sequenz und Weiterleitung der daraus resultierenden vermittlungstechnische Ereignisse VMTE (z.B. Tln(aktiv), Tln(passiv), Tln(Wahlziffer n), Tln(Flash)) an die CC-Funktion.
- Die ankommenden vermittlungstechnischen Ereignisse von der CC-Funktion (z.B. Tln(Ruf i), Tln(Wählzeichen), Tln(Besetztzeichen) in logische Signale zu wandeln und diese zeit- und sequenzgerecht der Konvertierungsfunktion bereit zustellen.

Während der Ruf über die Variable r gesteuert wird, erfolgt die Ausgabe von Hörzeichen an den Teilnehmer über den NF-Kanal, d.h. durch sequenzgerechte Programmierung der Empfangsseite des Codecs auf eine solche Zeitlage, auf der ein Hörton (z.B. 400 Hz) angeboten wird. In Abbildung 14c und Abbildung 2 Ist die Anschaltung des Hörtongenerators an das PCM-Koppelfeld angedeutet.

4 Funktionalität des Moduls SIG

Im Abschnitt 3 wurden die Grundzusammenhänge der a/b-Signalgabe behandelt. In diesem Abschnitt sollen, ausgehend von der Hard-Software-Struktur (Abbildung 2) die Funktionalität des Programmes SIG und das Zusammenwirken mit dem Programm CC (Call Control) näher betrachtet werden.

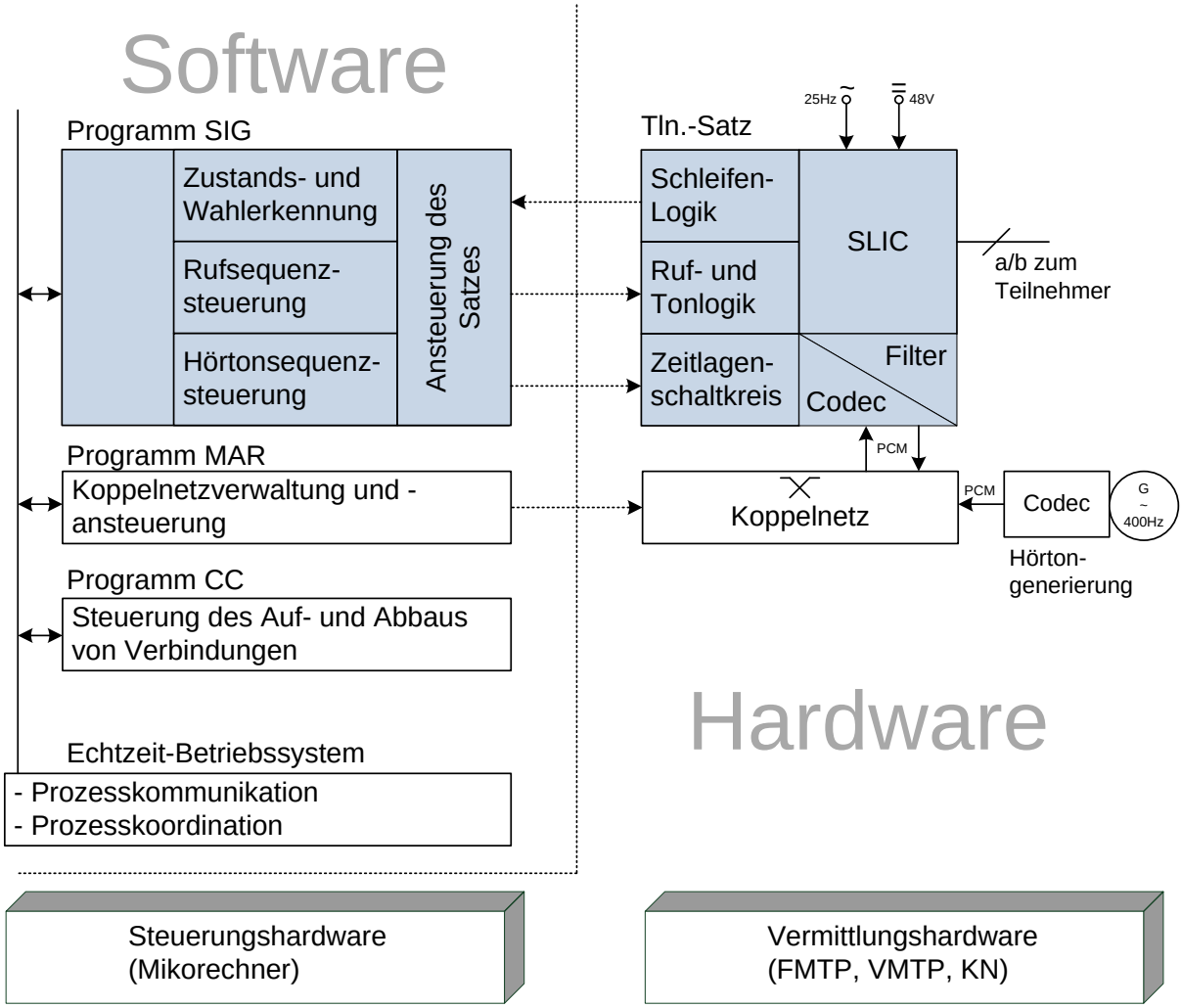


Abbildung 2: Hard-Software-Struktur der a/b-Signalgabe

In diesem Bild sind neben der Signalgabehard- und –software (farbige Teile) auch die Umgebungen angedeutet.

4.1 Hardware

Die Steuerungshardware besteht im Allgemeinen aus einem oder mehreren Mikrorechnern, die untereinander über Interfaces verbunden sind. Auf die Teilnehmersatzhardware wurde ausreichend im Abschnitt 3 eingegangen.

4.2 Software

Die Steuerung einer Vermittlung muss in Echtzeit erfolgen. Dies ist nur mit einem speziellen Programm, einem Echtzeitbetriebssystem, möglich. Dies hat in der Regel die Aufgaben:

- Organisation und Steuerung des Meldungsaustausches zwischen den Prozessen SIG, MAR, CC. Diese Aufgabe bezeichnet man als Prozesskommunikation.
- Organisation des Prozessablaufes, d.h. Starten, Stoppen, Unterbrechen usw. von Programmen zur Sicherung des Echtzeitbetriebes. Diesen Teil bezeichnet man als Prozesskoordination.

Das Programm CC steuert den Auf- und Abbau von Verbindungen. Beispielhafte Abläufe sind im Arbeitsblatt „SDL“ und im Praktikumsversuch „SDL 4: Entwurf, Simulation einer einfachen PABX mittels Postmasterinterface“. Die CC-Funktion nimmt dazu von der SIG-Funktion Meldungen über Tln-Zustandsänderungen entgegen, bewertet diese und gibt entsprechende Meldungen an ihre Umgebung (SIG, MAR, …) ab.
Die Meldungsinterfaces sind in Abbildung 3 dargestellt.

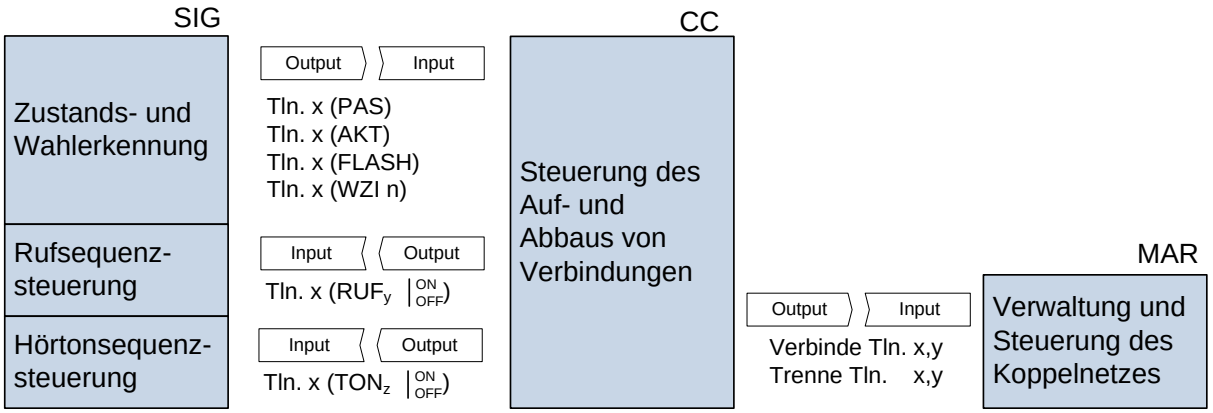


Abbildung 3: Meldungsinterfaces SIG-CC, CC-MAR

Im folgenden Abschnitt werden die einzelnen Programmteile:
Zustands- und Wahlerkennung, Rufsequenzsteuerung, Hörtonsequenzsteuerung näher betrachtet.

5 Programmodul SIG

5.1 Zustands- und Wahlerkennung

5.1.1 Grundlagen

Gemäß Abschnitt 3 setzt sich die Zustands- und Wahlerkennung aus der physikalisch-logischen Konvertierungsfunktion und der Signalisierungsfunktion, die die Bewertung der logischen Signale vornehmen, zusammen. Teilnehmerzustände und Wahlkennzeichen ergeben sich aus der zeitlichen Folge von Schleifenzuständen.

Abbildung 4 zeigt den Zusammenhang zwischen Schleifenzuständen und vermittlungstechnischen Ereignissen.

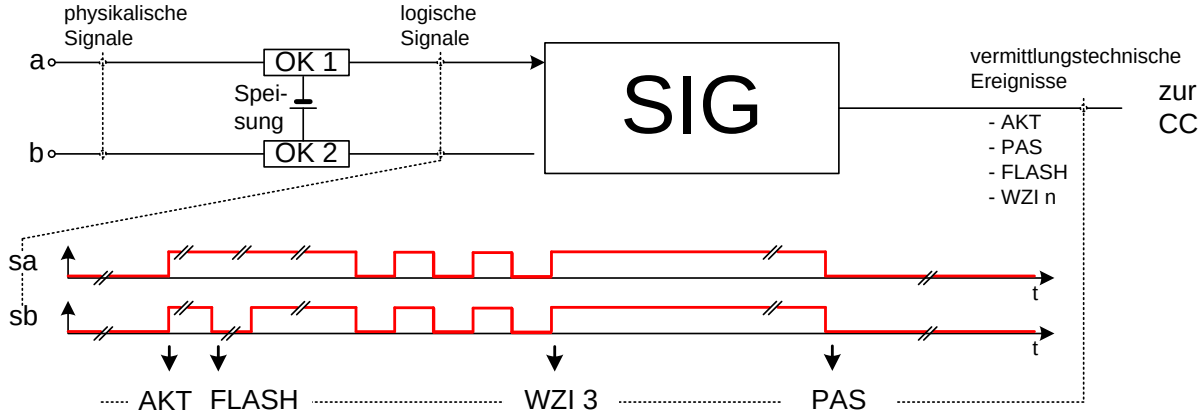


Abbildung 4: Zusammenhang zwischen logischen Schleifenzuständen und vermittlungstechnischen Ereignissen

Die zeitlichen Forderungen zur Auswertung der Schleifensignale ergeben sich aus Festlegungen zur Nummernschalterwahl. Danach ist für die Dauer t_a eine Wahlimpulses festgelegt:

90 ms =< t_a =< 110 ms.

Das Verhältnis k von Schleifenunterbrechungszeit t_u zu Schleifenschließzeit t_s ist festgelegt zu: 1,3 =< (k=t_u / t_s) =< 1,9.

Daraus ergibt sich das in Abbildung 5 dargestellte Toleranzviereck für Nummernschalterwahl. Alle Wertepaare (t_u, t_s), die innerhalb dieses Vierecks liegen, sind gültige Wahlimpulse.

$$\hat{t}_u = \frac{\hat{t}_a \cdot \hat{k}}{\hat{k}+1} = 72ms \qquad \hat{t}_s = \frac{\hat{t}_a}{\hat{k}+1} = 48ms$$

$$\check{t}_u = \frac{\check{t}_a \cdot \check{k}}{\check{k}+1} = 51ms \qquad \check{t}_s = \frac{\check{t}_a}{\check{k}+1} = 31ms$$

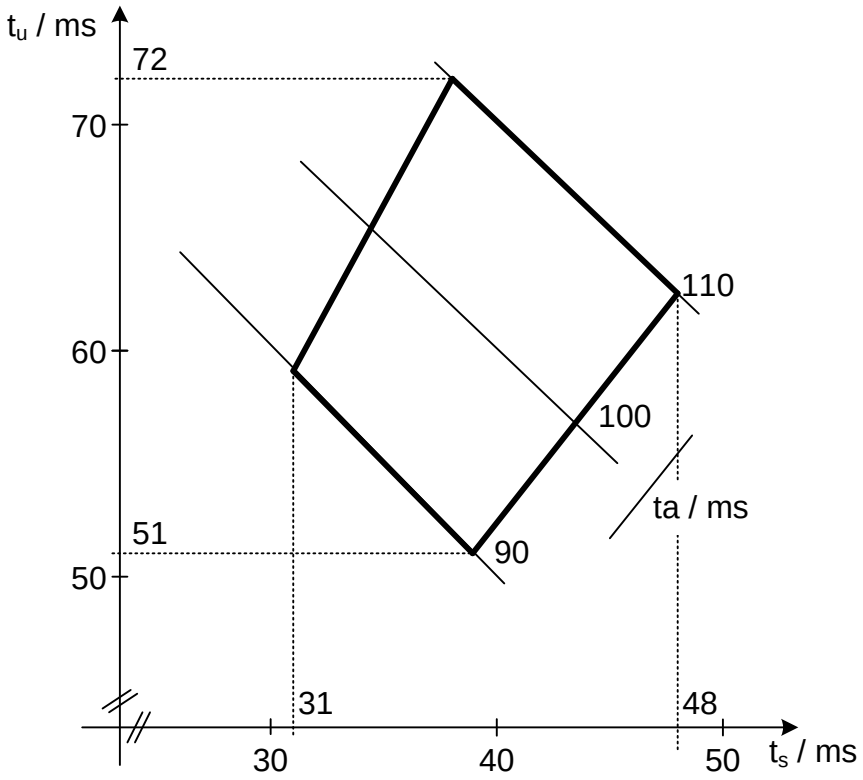


Abbildung 5: Toleranzschema für Nummernschalterwahl

Schleifenzustände können durch zwei Verfahren bewertet werden:

1. Ereignisgesteuerte Methode:
   Bearbeitungsauslösung durch Schleifenzustandsänderung und Ausmessung der Zeit bis zur nächsten Änderung.
2. Abtastmethode:
   Zyklische Abtastung der Schleife in einem hinreichend kleinen Zeitraster und Verarbeitung von Änderungen.

Für beide Verfahren gilt, dass eine zeitliche Bewertung der Schleifenzustände nur so genau sein kann, wie der Abstand zwischen zwei Abtastungen bei Methode 2 bzw. die Zeitquanten bei Methode 1.

Die ereignisgesteuerte Methode hat den Vorteil, dass der Rechner nur dann belastet wird, wenn Schleifenänderungen auftreten.

Der Nachteil ist, dass eine spezielle, interruptauslösende Peripherie erforderlich ist. Deshalb wird überwiegend die Abtastmethode angewendet.

5.1.2 Festlegung der Bearbeitungsperioden

Bei der 2. Methode werden die logischen Schleifensignale (s_a,s_b) zyklisch abgetastet, die Zustände mit dem vorhergehenden Abtastwert verglichen und bei Änderungen entsprechende Zustandsübergänge abgespeichert (Abbildung 6).

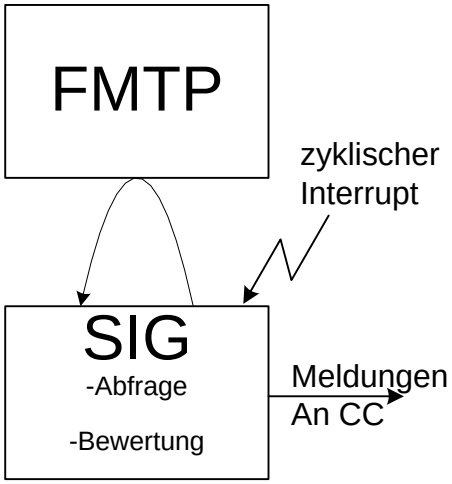


Abbildung 6: Prinzip der Abtastmethode

Bei einer solchen zeitdiskreten Bewertung treten Flankenunsicherheiten dT auf, deren Maximalwert durch die Abtastperiode T_B gegeben ist. Die maximale Flankenunsicherheit dT_{max} beträgt: $dT_{max}=T_B$.

Die Impulsverzerrung B, als Ergebnis der Abtastung, berechnet sich aus: $B<=T_B/t_1$, wobei t_1 die Impulsdauer ist.

Die Wahl der Bearbeitungsperiode ist damit ein Kompromiss zwischen ausreichend zeitlicher Auflösung der abzutastenden Signale und der Rechnerbelastung. Als Minimum für eine sichere Zustandserkennung sind zwei Bewertungen anzunehmen. Die größte zulässige Abtastperiode T_{Bmax} ist von der kleinsten auszumessenden Zeit abhängig. Aus dem Toleranzschema (Abbildung 5) entnimmt man: t_{smin}=31ms.

Daraus folgt: $T_{Bmax}=t_{smin}/2$=15.5ms.

Üblich sind Bearbeitungsperioden von 6 - 12ms (NZ400=15ms). Hier sollen 10ms festgelegt werden.

5.1.3 Automatenstruktur

Wie bereits erwähnt, ist für die Realisierung dieser Aufgabe eine „Schaltung mit Gedächtnis“ erforderlich, da ja der vorhergehende Abtastzustand mit dem aktuellen verglichen werden muss. Solche Schaltungen bezeichnet man als Automat. Für die programmtechnische Realisierung wird eine Moore-Struktur nach Abbildung 7 zugrundegelegt.

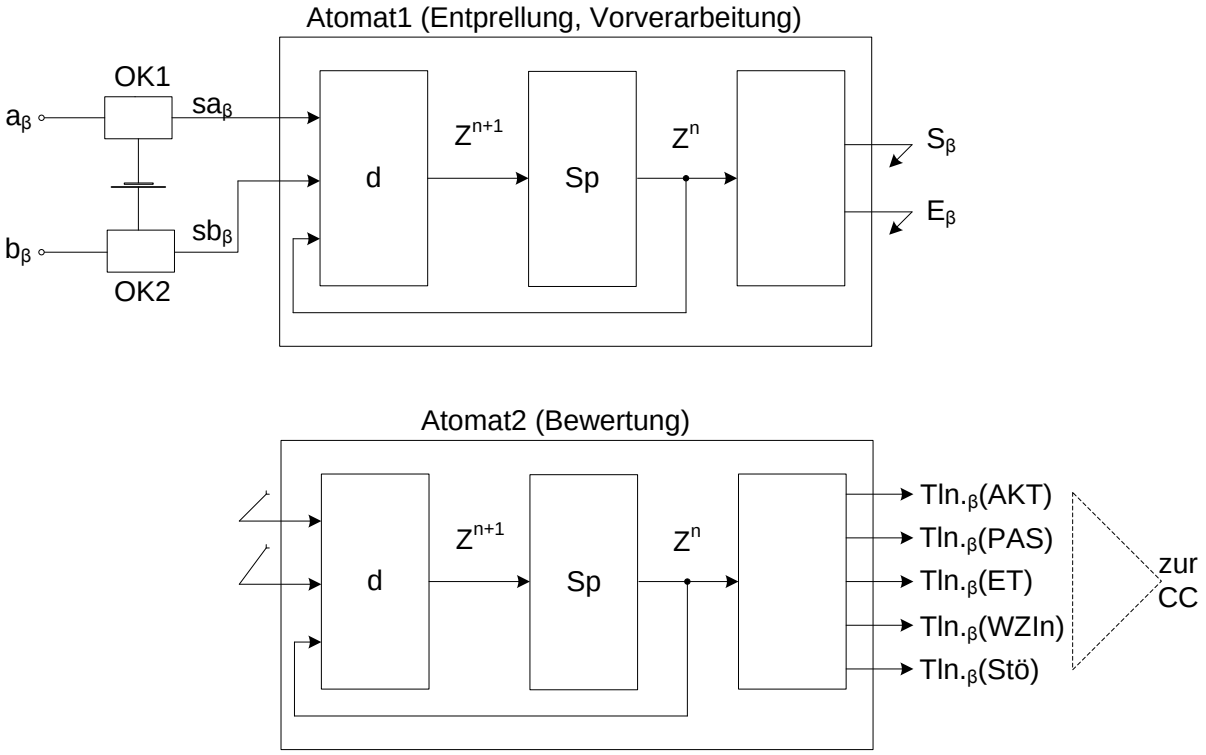


Abbildung 7: Automatenstruktur zur Zustands- und Wahlerkennung

Für beide Automaten gilt: Z^{n+1} =delta (X^n,Z^n),

Y^n =lambda(Z^n),

mit:

x	=Eingangswort,
y	=Ausgabewort,
Z	=Zustandswort,
delta	=Übergangsvorschrift,
lambda	=Ausgabevorschrift.

D.h., dass der Folgezustand des Automaten Z^{n+1} vom momentanen Zustand Z^n und dem anliegenden Eingangswort X^n abhängt.

Das Ausgabewort Y ist nur vom Zustand Z des Automaten abhängig.

5.1.3.1 Automat zur Entprellung und Vorverarbeitung

Bei Verwendung mechanischer Nummernschalter kann es zu Kontaktprellungen kommen. Diese sollen durch den Automaten unterdrückt werden. Insgesamt ergeben sich folgende Aufgabenstellungen:

- Entprellung für sa entsprechend der Vorschrift in Abbildung 8
- Bildung der Variablen E_β: $E_\beta^n=S_\beta^n$ und $S_{b\beta}^n$.

Abbildung 8 illustriert, wie der Automat funktionieren soll. Gezeigt werden die gestörten logischen Signale $s_{a\beta}$, $s_{b\beta}$ und die gewünschten Ergebnisse s_β^n und E_β^n bei einem Abtasttakt von 10ms.

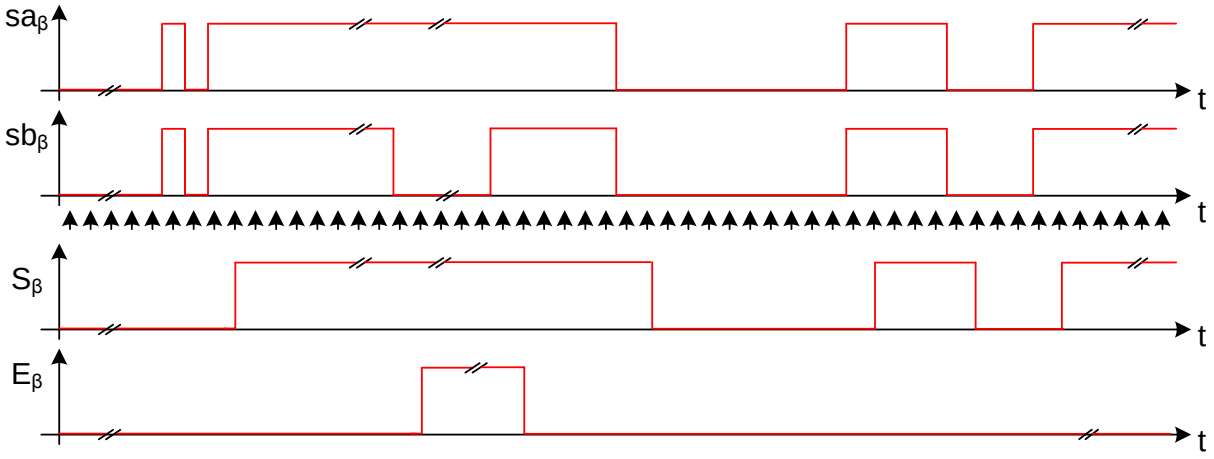
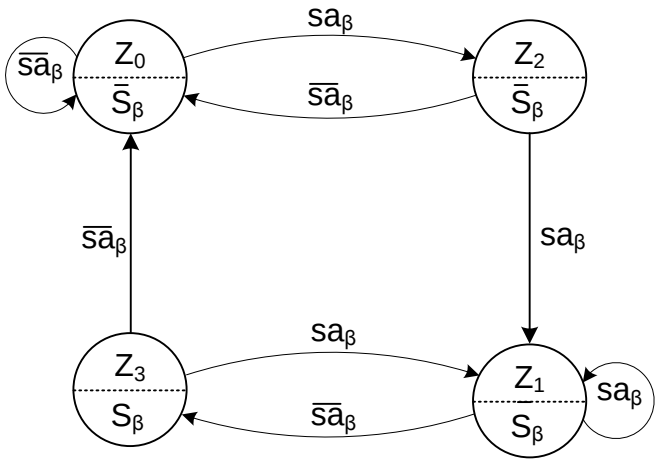


Abbildung 8: Aufgabenstellung für Automat1

Für die Entprellung des Schleifensignals $s_{a\beta}$ lässt sich folgender Automatengraph bzw. folgende Übergangstabelle aufschreiben (Abbildung 9).

Die Codierung der Zustände wurde darin so gewählt, damit die Ausgangsbuchstaben durch das niederwertigste Bit des Zustandswortes gebildet werden können.



Z^n \ Z^{n+1}	sa_β	$\overline{sa}_\beta$	
Z_0	Z_2	Z_0	$\overline{S}_\beta$
Z_1	Z_1	Z_3	S_β
Z_2	Z_1	Z_0	$\overline{S}_\beta$
Z_3	Z_1	Z_0	S_β

Abbildung 9: Graph und Tabelle des Automaten

5.1.3.2 Automat zur Zustandserkennung und Wahlbewertung

Der in Abbildung 9 gezeigte Graph ist für eine Abtastperiode von 10ms entworfen. Damit kann man die Einhaltung des Toleranzschemas (Abbildung 5) nicht exakt überprüfen. Dies ist bei elektronischen Vermittlungsanlagen aber auch nicht erforderlich. Hier kann man größere NUS-Toleranzen zulassen, da keine elektromechanischen Baugruppen (Relais, Wähler) angesteuert werden.

Aus Abbildung 7 erkennt man, dass die Eingangsbuchstaben durch die Variablen S_β, E_β gebildet werden und der Automat die Ausgangsbuchstaben Tln_β(AKT), Tln_β(PAS),Tln_β(FLT), Tln_β(WZIn), Tln_β(STÖ) ausgeben soll. Der Automat hat insgesamt 56 Zustände.

Hauptzustände und ihre Bedeutung sind:

- 00 Hörer aufgelegt,
- 04 Hörer abgenommen,
- 36 Wahlserienende,
- 38 Teilnehmer ist gestört

In folgenden Zuständen werden Meldungen an die CC-Funktion übergeben:

- 50 Teilnehmer ist aktiv (AKT),
- 51 Teilnehmer hat mit der Wahl begonnen (WBG),
- 52 Teilnehmer ist passiv geworden (PAS)
- 53 Teilnehmer hat Flash-Taste gedrückt (FLT)
- 54 Teilnehmer hat Wahlziffer n gewählt (WZIn),
- 55 Teilnehmer ist gestört (STÖ).

Die Meldungszustände haben Zustandsbezeichner von 50 bis 55. Diese Codierung wurde gewählt, damit man die Meldungszustände bei der programmtechnischen Umsetzung einfach ermitteln kann (if(Zustand >= 50){ Meldung }).
Der Initialzustand des Automaten ist der Zustand 0. Nach Abheben des Hörers geht der Automat über die Zustände 1,2,3,50 in den Zustand 4.
Beim Wahlbeginn werden die Zustände 51,5-10 im t_u-Zweig durchlaufen. Die Zustände 6 bis 10 bilden das Zeitfenster für die Zeit tu, d.h. alle Zeiten von 30ms bis 70ms sing gültig. Mit der Eingangsbelegung S_β in einem dieser Zustände wird der t_s-Zweig erreicht (Zustände 40, 21,…). In diesem Zweig bilden die Zustände 22 bis 25 das Zeitfenster, d.h., dass alle Zeiten von 30ms bis 60ms gültige t_s-Zeiten sind.
Ist t_s größer, wird über die Zustände 26 bis 54 der Zustand 36 (Wahlserienende) erreicht. Im Zustand 54 wird die Wahlziffer ausgegeben und Wahlimpulszähler zurückgesetzt. Dieser Zähler ist eine Variable, die jedesmal beim Durchlaufen des Zustandes 56 inkrementiert wird.
Auf der rechten Seite der Abbildung 10 ist der Störungszweig zu finden. Dieser Zweig wird immer dann erreicht, wenn der NUS nicht in den programmierten Zeitfenstern liegt.

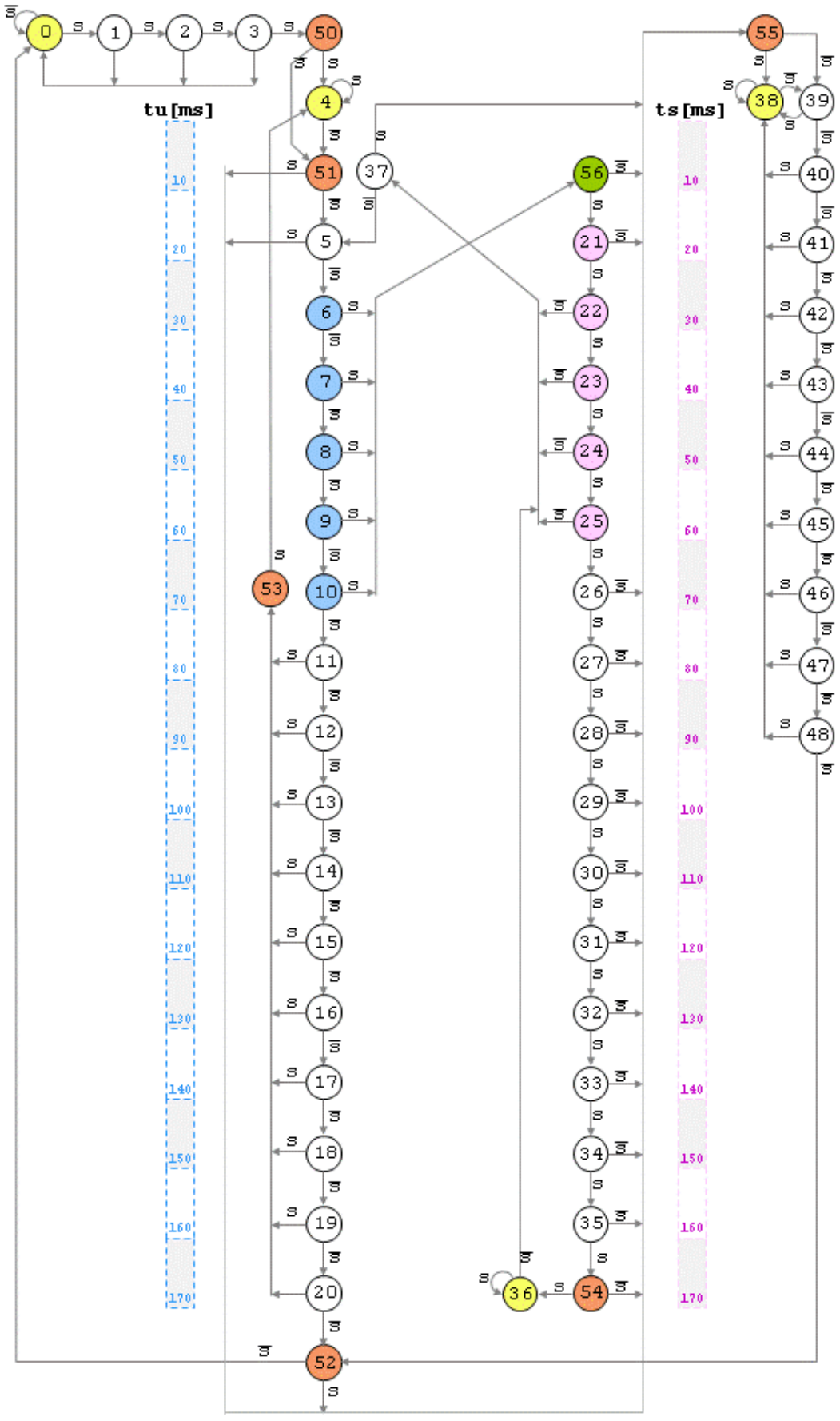


Abbildung 10: Graph des Automaten zur Zustandserkennung und Wahlbewertung

5.1.4 Softwarerealisierung

Die im Abschnitt 5.1.3 besprochene Automatenstruktur

A=A(X,Z,Y,delta,lambda)

und deren mathematische Beschreibung gilt nur dann, wenn die Zustandsübergänge getaktet erfolgen. Das heißt, es wird vorausgesetzt, dass die Verarbeitung von Eingangszuständen zu diskreten Zeitpunkten erfolgt und die Bildung der entsprechenden inneren Zustände und Ausgangszustände bis zum nächsten Verarbeitungszeitpunkt abgeschlossen sind. Realisiert man einen solchen Automaten durch Software, d.h. durch eine Sequenz von Befehlen, ist eine Erweiterung des Taktbegriffes entsprechend Abbildung 11 erforderlich.

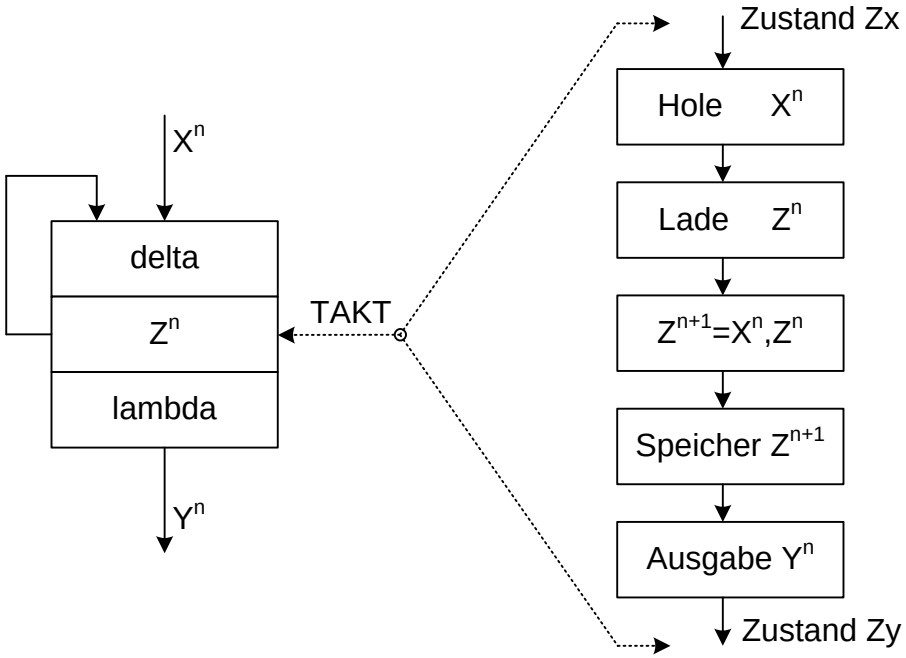


Abbildung 11: Taktinterpretation für Hard- und Softwareautomaten

Bei einer Softwarerealisierung müssen die in einem Hardwareautomaten parallel ablaufenden Vorgänge seriell abgewickelt werden. Der diskrete Zeitpunkt der Wirkung des Taktes beim Hardwareautomaten wird zur diskreten Übergangszeit beim Softwareautomaten.
Die Umsetzung des Automaten 2 in ein C++-Programm ist in folgendem Quellcode dargestellt. Ersichtlich wird daraus, dass wichtige Variablen in der Header-Datei der Hauptklasse „PbxCC.h“ deklariert wurden. Dies ist notwendig, um diese Variablen auch anderen Klassen zur Verfügung stellen zu können. Hierdurch bedingt, muss die Klasse PbxSigAB in der Hauptklasse per Forward-Deklaration bekannt gemacht werden.
Das mehrdimensionale Array „DeltaAutomat2“ repräsentiert den Automat zur Zustandserkennung und Wahlbewertung, wobei der Index 1 den aktuellen Zustand und Index 2 (die jeweils in geschweiften Klammern stehenden Zahlenpaare) die Folgezustände für $\overline{\text{Schleife}}$ und Schleife bezeichnen. Zur Kommunikation der einzelnen Module untereinander wird die Struktur „header_K“ verwenden.

MyCC.h (Auszug aus der Header-Datei der Hauptklasse)

```
// Automat zur Zustands- und Wahlerkennung
int DeltaAutomat2[57][2]={
/* 0   */             {0,1},{0,2},{0,3},{0,50},{51,4},
/* 5   */             {6,55},{7,56},{8,56},{9,56},{10,56},
/* 10  */             {11,56},{12,53},{13,53},{14,53},{15,53},
/* 15  */             {  ,  },{  ,  },{  ,  },{  ,  },{  ,  },
/* 20  */             {  ,  },{  ,  },{  ,  },{  ,  },{  ,  },
/* 25  */             {  ,  },{  ,  },{  ,  },{  ,  },{  ,  },
/* 30  */             {  ,  },{  ,  },{  ,  },{  ,  },{  ,  },
/* 35  */             {  ,  },{  ,  },{  ,  },{  ,  },{  ,  },
/* 40  */             {  ,  },{  ,  },{  ,  },{  ,  },{  ,  },
/* 45  */             {  ,  },{  ,  },{  ,  },{  ,  },{  ,  },
/* 50  */             {  ,  },{  ,  },{  ,  },{  ,  },{  ,  },
                      {  ,  },{  ,  }};

// Struktur zum Versenden von Nachrichten
struct header_K{
       char dest;            /*Empfänger z.B. C für Call Control*/
       char sour;            /*Absender z.B. P für Prozess Manager*/
       messages requ;        /*Meldung z.B. BZ*/
       int add1;             /*Adresse 1 z.B. 1 für Teilnehmer 1*/
       int add2;             /*Adresse 2 z.B. 2. für Teilnehmer 2*/
       tlnRole role;         /*Tln-Role (a,b,f)*/
       int time;             /**/
};

// Forward-Deklaration
class TPbxSigAB;
```

SIGAB.h

```
//---------------------------------------------------------------------------
#ifndef PbxSigABH
#define PbxSigABH
//---------------------------------------------------------------------------
#endif

#include <Dialogs.hpp>
#include "PbxCC.h"


class TPbxSigAB{
private:
       int SchleifenProbe; //0;
       int ImpulsZaehler[4]; //{0,0,0,0};
       int Schleife[4]; //{0,0,0,0};
       TForm1 *PbxCC;

public:
       header_K s_header_S[4];
       int ZustandAutomat2[4]; //{0,0,0,0};

       __fastcall TPbxSigAB(void *CC);

       void __fastcall SpeicherSchleife(char buffer[6]);
       void __fastcall SchleifenBewertung();
protected:
};
```

SIGAB.cpp

```
/****************************** INFO SIGAB *********************************/
/*Funktion
                                        */
/* - Abtasten der Schleifenzustände einer Teilnehmerkarte aller 10ms              */
/* - Bewertung der Schleifenzustände (Automat2)
      */
/***************************************************************************/

//---------------------------------------------------------------------------
#pragma hdrstop

#include "PbxSigAB.h"
```

```
//---------------------------------------------------------------------------

#pragma package(smart_init)

__fastcall TPbxSigAB::TPbxSigAB(void *CC){
	// CC-Objekt anlegen
	PbxCC = (TForm1*) CC;

	// Initialisierung
	SchleifenProbe=0x00;
	for (int i=0; i<4; i++) {
		ZustandAutomat2[i]=0x00;
		ImpulsZaehler[i]=0;
		Schleife[i]=0x00;
		s_header_S[i].requ=pas;
		s_header_S[i].add1=i;
	}
}

void __fastcall TPbxSigAB::SpeicherSchleife(char buffer[6]){
	switch (buffer[1]) {
	case 0x53 :
	 switch (buffer[2]) {
		case 0x01:
			char schleife=buffer[3];
			for (int i=0; i<4; i++) {
				// Schleife für einzelne Teilnehmer speichern
				Schleife[i]=schleife&0x01;
				// Bitverschiebung um eine Stelle nach rechts
				schleife=schleife>>1;
			}
			// Bewertung der Schleife
			SchleifenBewertung();
			break;
		}
		break;
	case 0x55 : // Taste wurde gedrückt (MFV-Wahl)
			ZustandAutomat2[(buffer[2]-0x30)]=54;
			switch (buffer[3]) {
			case 0x11: s_header_S[(buffer[2]-0x30)].requ=wz1;break;//0x11
			case 0x12: s_header_S[(buffer[2]-0x30)].requ=wz2;break;//0x12
			case 0x13: s_header_S[(buffer[2]-0x30)].requ=wz3;break;//0x13
			case 0x14: s_header_S[(buffer[2]-0x30)].requ=wz4;break;//0x14
			case 0x15: s_header_S[(buffer[2]-0x30)].requ=wz5;break;//0x15
			case 0x16: s_header_S[(buffer[2]-0x30)].requ=wz6;break;//0x16
			case 0x17: s_header_S[(buffer[2]-0x30)].requ=wz7;break;//0x17
			case 0x18: s_header_S[(buffer[2]-0x30)].requ=wz8;break;//0x18
			case 0x19: s_header_S[(buffer[2]-0x30)].requ=wz9;break;//0x19
			case 0x1A: s_header_S[(buffer[2]-0x30)].requ=wz0;break;//0x1A
			}
			s_header_S[(buffer[2]-0x30)].dest='P';
			s_header_S[(buffer[2]-0x30)].sour='S';
			s_header_S[(buffer[2]-0x30)].add1=(buffer[2]-0x30);
			s_header_S[(buffer[2]-0x30)].add2=0;
			// Message senden
			PbxCC->SendK(&s_header_S[(buffer[2]-0x30)]);
		break;
	}
}

void __fastcall TPbxSigAB::SchleifenBewertung(){
	for (int z=0; z<4; z++) {
		// aktuellen Zustand, abhängig von alten Zustand und Schleife speichern
		ZustandAutomat2[z]=DeltaAutomat2[ZustandAutomat2[z]][Schleife[z]];
		if (ZustandAutomat2[z] == 56) {
			// Anzahl der Impulse zählen ≙ gewählter Ziffer
			ImpulsZaehler[z]++;
		}
		if (ZustandAutomat2[z] >= 50 && ZustandAutomat2[z] != 56) {
			 s_header_S[z].dest='P';
			 s_header_S[z].sour='S';
			 switch (ZustandAutomat2[z]) {
			 case 50:
				s_header_S[z].requ=akt;break;
			 case 51:
				s_header_S[z].requ=wbg;break;
			 case 43:
				s_header_S[z].requ=et1;break;
```

```
                        case 52:
                              s_header_S[z].requ=pas;break;
                        case 46:
                              s_header_S[z].requ=et2;break;
                        case 55:
                              s_header_S[z].requ=sto;break;
                        }
                        if (ZustandAutomat2[z]==54) {// Ziffer wurde gewählt (IWV)
                              switch (ImpulsZaehler[z]) {
                              case 10:
                                    s_header_S[z].requ=wz0;break;
                              case 1:
                                    s_header_S[z].requ=wz1;break;
                              case 2:
                                    s_header_S[z].requ=wz2;break;
                              case 3:
                                    s_header_S[z].requ=wz3;break;
                              case 4:
                                    s_header_S[z].requ=wz4;break;
                              case 5:
                                    s_header_S[z].requ=wz5;break;
                              case 6:
                                    s_header_S[z].requ=wz6;break;
                              case 7:
                                    s_header_S[z].requ=wz7;break;
                              case 8:
                                    s_header_S[z].requ=wz8;break;
                              case 9:
                                    s_header_S[z].requ=wz9;break;
                              }
                        }
                        ImpulsZaehler[z]=0;
                        s_header_S[z].add1=z;
                        PostMessage(Application-
>Handle,WM_PbxMessage,0,(LPARAM)&s_header_S[z]);
                  }
            }
}
```

5.2 Hörton- und Rufsteuerung

5.2.1 Grundlagen

Bevor auf die technische Realisierung der Ruf- und Hörtonsteuerung eingegangen wird, ist es erforderlich die grundlegenden Festlegungen zu erörtern. In der CCITT-Empfehlung Q.35 (Hörzeichen für nationale Zeichengabesysteme) werden die Regelungen getroffen, die für internationale Verbindungen erforderlich sind. Dies betrifft die Tonfrequenzen und die Sequenzen für das Frei- und Besetztzeichen. Festlegungen zu anderen Bediensignalen sind Sache der nationalen Verwaltungen. Für die Tonfrequenzen ist ein Bereich von 300Hz bis 475Hz zulässig wobei Frequenzen zwischen 400Hz und 450Hz empfohlen werden.
Die Verhältnisse von Signalzeit zu Pausenzeit (Sequenz) für das Besetzt- und Freizeichen gehen aus Abbildung 12 hervor.

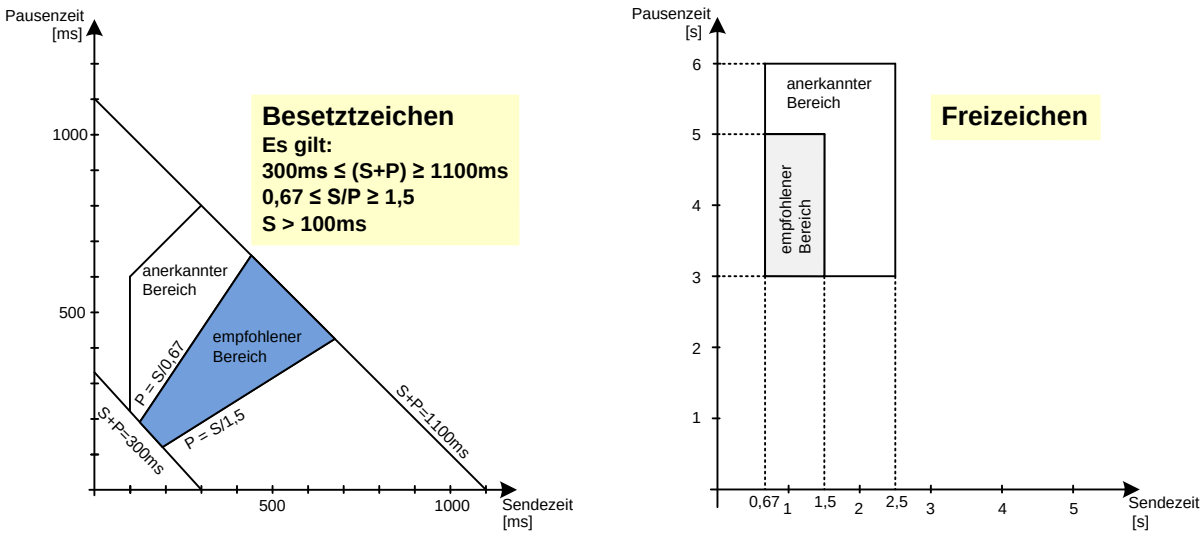


Abbildung 12: Zeitbedingungen für das Besetzt- und Freizeichen nach Q.35

In modernen Vermittlungsanlagen, insbesondere Nebenstellenanlagen, sind darüber hinaus weiter Hör- und Rufzeichen sinnvoll und erforderlich. Einige werden nachfolgend genannt und ihre Verwendung erläutert:

- Rufzeichen (RZ)
  - Ankündigung eines kommenden Gesprächs. In Nebenstellenanlagen kann damit ein Interngespräch gekennzeichnet werden.
- Sonderrufzeichen (SRZ)
  - In Nebenstellenanlagen können damit Externgespräche angekündigt werden.
- Anklopfzeichen (AKZ)
  - Signalisierung, dass weiteres Gespräch ansteht.
- Aufschaltezeichen (ASZ)
  - Signalisierung, dass ein weiterer Teilnehmer in eine Gesprächsverbindung eintritt.
- Besetztzeichen (BZ)
- Freizeichen (FZ)
- Negativquittungszeichen (NQZ)
  - Ablehnung von angeforderten Erleichterungen (facilities) in Nebenstellenanlagen.
- Positivquittungszeichen (PQZ)
  - Kann in Nebenstellenanlagen zur Quittierung von angeforderten Erleichterungen (z.B. Rufumleitung) verwendet werden.
- Sonderwählzeichen (SWZ)
  - Kann in Nebenstellenanlagen zur Signalisierung dafür verwendet werden, dass der Teilnehmer durch eine aktivierte Erleichterung zur Zeit nicht erreichbar ist.

- Wählzeichen (WZ).

Im Hinblick auf die Erzeugung dieser Kennzeichen ist es sinnvoll, ihnen einen gemeinsamen zeitlichen Bezug zu geben. Dies bezieht sich sowohl auf die Periode, als auch auf die Zeitquanten. In Abbildung 13 sind für einige Ruf- und Hörzeichen mögliche Sequenzen dargestellt. Dabei wurden für die Periode aller Kennzeichen eine Zeit von 5000ms und für ein Zeitquant 250ms festgelegt.

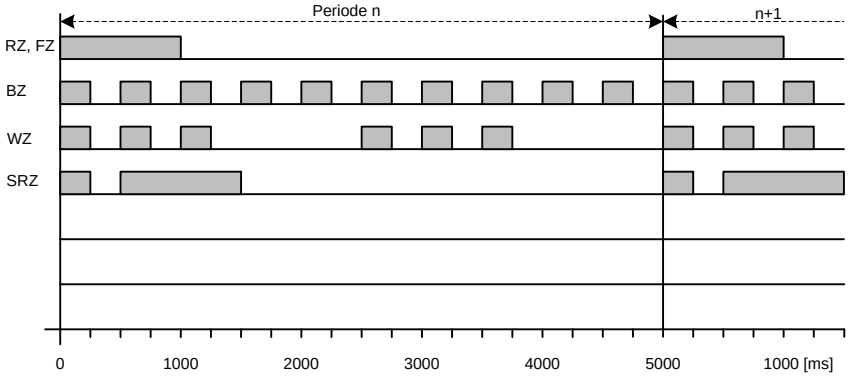


Abbildung 13: Steuersequenzen für ausgewählte Kennzeichen

5.2.2 Realisierung der Hörton- und Rufsteuerung

Wie aus Abbildung 3 und Abbildung 14 hervorgeht und eingangs besprochen wurde, besteht eine Vermittlung aus speziellen Hardwarekomponenten die über Interfaces (PIO, SIO, UART, spezielle) angesprochen werden können.

Im Falle der Rufsteuerung muss über ein Interface die Variable r mit HIGH belegt werden damit das Relais R anzieht und über den Kontakt r_1 der Rufgenerator an die a/b-Adern angeschaltet wird. Führt die Variabel r die Belegung LOW, fällt das Relais ab und der Ruf wird abgeschaltet.

Das Programm zur Rufsteuerung hat damit die Aufgabe, zu Beginn jeder Rufperiode an die Teilnehmerschaltungen deren Teilnehmer gerufen werden sollen, die Variable r mit HIGH zu belegen. Nach beispielsweise 1000ms ist die Belegung rückgängig zu machen. Man erkennt, dass beliebige Sequenzen und damit Rufarten softwaremäßig ohne Hardwareeingriffe realisierbar sind.

Für die Hörtonerzeugung trifft ähnliches zu. Die Hardwarevoraussetzung sind ebenfalls in Abbildung 2 und Abbildung 14 angedeutet. Im einfachsten Fall verwendet man für jeden Hörton (400Hz, 500Hz oder andere) einen Generator. Vermittels von Codecs werden die Hörtöne pulscodemoduliert und über das PCM-Koppelfeld an die Teilnehmergruppen verteilt. Auf den PCM-Highways der Gruppen belegen sie in der Regel feste Zeitlagen (z.B. 400Hz auf Zeitlage 29, 500Hz auf Zeitlage 30 usw.). soll nun einem oder mehreren Teilnehmern ein Hörton aufgeschaltet werden, erfolgt dies durch sequenzrichtiges Programmieren der Codec-Empfangsseiten auf die Zeitlage, auf der die gewünschte Frequenz liegt.

Ein Programm zur Ruf- und Hörtonsequenzsteuerung ist in folgendem Quellcode dargestellt. Dieses Programm wird aller 250ms durch das Betriebssystem aufgerufen. Während beim Programm zur Zustands- und Wahlbewertung nur ein Teilnehmer betrachtet wurde, soll hier von einer kleinen Nebenstellenanlage mit 4 Teilnehmern ausgegangen werden. Es sei darauf verwiesen, dass dieses Programm nur die Funktionalität verdeutlichen soll und nicht optimiert ist. Wichtige Konstanten und Typen wurden wieder in der Header-Datei der Hauptklasse definiert. Hierdurch bedingt, wurde die Klasse TPbxSigHT per Forward-Deklaration der Hauptklasse bekannt gemacht. Das mehrdimensionale Array SequenzTabelle beinhaltet hierbei die Sequenzen zur Darstellung der einzelnen Ruf- und Hörtöne.

PbxCC.h (Auszug aus der Header-Datei der Hauptklasse)

```
typedef enum {E,A,U} RufHoerZeichenZustand; //E=Ein, A=Aus, U=Unverändert

// Ruf- und Hörtöne definieren
const RufHoerZeichenZustand SequenzTabelle[11][20] = {
/*NO */{A,A,A,A,A,A,A,A,A,A,A,A,A,A,A,A,A,A,A,A},
/*BZ */ {E,A,E,A,E,A,E,A,E,A,E,A,E,A,E,A,E,A,E,A},
/*FZ */ {E,U,U,U,A,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U},
/*WZ */ {E,A,E,A,E,A,U,U,U,U,E,A,E,A,E,A,U,U,U,U},
/*PQZ*/ {E,U,U,U,U,U,A,U,U,U,U,U,U,U,U,U,U,U,U,U},
/*NQZ*/ {E,A,E,A,E,A,E,A,E,A,E,A,E,A,E,A,E,A,E,A},
/*AKZ*/ {A,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U},
/*RZ */ {E,U,U,U,A,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U},
/*SRZ*/ {E,A,E,U,A,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U},
/*ERZ*/{E,U,U,U,A,U,U,U,E,U,U,U,A,U,U,U,E,U,U,U},
/*EFZ*/{E,U,U,U,A,U,U,U,E,U,U,U,A,U,U,U,E,U,U,U}};

typedef enum {
/*Sig_ab an CC  */    pas,akt,et1,et2,sto,wbg,d01,d02,d03,d04,
                      wz0,wz1,wz2,wz3,wz4,wz5,wz6,wz7,wz8,wz9,
/*CC an Ruf_hör */    no ,bz ,fz ,wz ,pqz,nqz,akz,rz ,srz,erz,efz,
/*CC an Time    */    bwb,cwb,bw ,cw ,bbz,cbz,brz,crz,d06,d07,
/*Time an CC    */    owb,ow ,obz,orz,d08,d09,d10,d11,d12,d13,
/*CC an Mar     */    con,dis,
/*Pro-Man an CC */    aAkt,aPas,aWzi,bAkt,bWzi,bPas,wrtTab,tWZ,tBZ,tRZ,
/*CC an Pro-Man */    kCC} messages;

class TPbxSigHT;
```

PbxSigHT.h

```
//---------------------------------------------------------------------------
#ifndef PbxSigHTH
#define PbxSigHTH
//---------------------------------------------------------------------------
#endif

#include <Dialogs.hpp>
#include "PbxAPI.h"
#include "PbxCC.h"


class TPbxSigHT{
private:
        int SequenzZaehler;
        int i;
        char Port[4];
        TPbxAPI *PbxAPI1;
        TForm1 *PbxCC;
        TTimer *interupt;

        RufHoerZeichenArt AuftragsTabelle[4];
```

```
public:
	char TonPort[4];
	char RufPort[4];


	__fastcall TPbxSigHT(TForm1 *CC, TPbxAPI *Pbx);
	__fastcall ~TPbxSigHT();

	void __fastcall AuftragEintragen(byte TlnNr,RufHoerZeichenArt Zeichen);
	void __fastcall TonEinAus(RufHoerZeichenArt k);
	void __fastcall RufEinAus(RufHoerZeichenArt l);
	void __fastcall Interrupt250ms(TObject *Sender);
	int __fastcall CountSequenzType(RufHoerZeichenArt k);
	void __fastcall PbxMessageSigHT(header_K headerR);

protected:
};
```

PbxSigHT.cpp

```
//---------------------------------------------------------------------------
#pragma hdrstop

#include "PbxSigHT.h"
//---------------------------------------------------------------------------

#pragma package(smart_init)


__fastcall TPbxSigHT::TPbxSigHT(TForm1 *CC, TPbxAPI *Pbx){
	// CC-Objekt anlegen
	PbxCC = CC;
	// Konstruktur - hier Initialisierungen vornehmen
	SequenzZaehler=0;
	// PBX-Verbindung vom Hauptprogramm übernehmen
	PbxAPI1 = Pbx;

	for (int u=0; u<4; u++) {
		AuftragsTabelle[u] = NO;
		// Port-Array initialisieren
		Port[u] = 0x30+u;
	}
	interupt = new TTimer(PbxCC);
	interupt->Interval=250;
	interupt->OnTimer=Interrupt250ms;
	interupt->Enabled;
}

__fastcall TPbxSigHT::~TPbxSigHT(){
	delete interupt;
}

void __fastcall TPbxSigHT::PbxMessageSigHT(header_K headerR){
	AuftragEintragen(headerR.add1,(RufHoerZeichenArt)headerR.requ-20);
}

void __fastcall TPbxSigHT::AuftragEintragen(byte TlnNr,RufHoerZeichenArt Zeichen){
	AuftragsTabelle[TlnNr]=Zeichen;
}

void __fastcall TPbxSigHT::TonEinAus(RufHoerZeichenArt k){
	switch(SequenzTabelle[k][SequenzZaehler]){
		case E:
			PbxAPI1->PbxTonAn(Port[i],PbxCC->GetReferenzNumber());
			break;
		case A:
			PbxAPI1->PbxTonAus(Port[i],PbxCC->GetReferenzNumber());
			break;
	}

}

void __fastcall TPbxSigHT::RufEinAus(RufHoerZeichenArt l){
	switch(SequenzTabelle[l][SequenzZaehler]){
		case E:
			PbxAPI1->PbxRufAn(Port[i],PbxCC->GetReferenzNumber());
			break;
```

```
                case A:
                        PbxAPI1->PbxRufAus(Port[i],PbxCC->GetReferenzNumber());
                        break;
        }
}

void __fastcall TPbxSigHT::Interrupt250ms(TObject *Sender){
        for (i=0; i<4; i++) {
                switch (AuftragsTabelle[i]) {
                        case NO:
                                TonEinAus(NO);
                                RufEinAus(NO);
                                break;
                        case BZ: TonEinAus(BZ);break;
                        case FZ: TonEinAus(FZ);break;
                        case WZ: TonEinAus(WZ);break;
                        case PQZ: TonEinAus(PQZ);break;
                        case NQZ: TonEinAus(NQZ);break;
                        case AKZ: TonEinAus(AKZ);break;
                        case RZ: RufEinAus(RZ);break;
                        case SRZ: RufEinAus(SRZ);break;
                        case ERZ: RufEinAus(ERZ);if(SequenzZaehler==19) AuftragsTabel-
le[i]=RZ;break;
                        case EFZ: TonEinAus(EFZ);if(SequenzZaehler==19) AuftragsTabel-
le[i]=FZ;break;
                }
        }
        SequenzZaehler=((SequenzZaehler+1)%20);
}

int __fastcall TPbxSigHT::CountSequenzType(RufHoerZeichenArt k){
        int counter=0;
        for (int i=0; i<4; i++) {
                if (AuftragsTabelle[i]==k) {
                        counter++;
                }
        }
        return counter;
}
```

6 Versuchsdurchführung

6.1 Allgemeines

Implementieren Sie eine einfache Call Control mit Zweierverbindungsszenarien. Nutzen Sie hierfür den in der Anlage beigefügten SDL-Entwurf.
Unter http://telecom.htwm.de/telecom/praktikum/html/v8.htm können Sie die zu implementierende Funktionalität anhand eines Beispielprogrammes testen.
Als Entwicklungsplattform wird Borland Developer Studio 2006 verwendet. Wie im 1. Versuchsteil kennengelernt, wird für die Kommunikation mit der Versuchshardware die C++-Komponente „PbxAPI“ verwendet. Die Komponente bietet eine Vielzahl von Methoden und Ereignissen zur Steuerung der Versuchshardware.
Das Programmgerüst zur Entwicklung der Call Control finden Sie unter http://telecom.htwm.de/telecom/praktikum/html/PABX/PbxV2.zip.
Dieses Programmgerüst beinhaltet bereits die fertige Oberfläche sowie die Klassen (Teilmodule) SigAB, SigHT und SigMar. Für die Entwicklung einer funktionierenden Call Control haben Sie die Aufgabe, die bereits angelegten Klassen (Teilmodule) PbxManager und PbxCallControl zu implementieren.

6.2 Die Kommunikation zwischen Modulen

Für die Kommunikation innerhalb der einzelnen Klassen (Teilmodulen) steht die Methode:

```
void __fastcall SendK(header_K *message);
```

zur Verfügung.

Die Methode „*SendK“* verschickt dabei die Nachrichten als Windows-Message. In der PbxCC-Methode „PbxMessage“ werden diese Nachrichten empfangen und an die entsprechenden Module verteilt. Um Nachrichten verschicken zu können, ist es zuvor notwendig diese in die vorgeschriebene Struktur *„header_K“* zu verpacken.

Der Aufbau von *„header_K“* lautet:

```
struct header_K{
       char dest;            /*Empfänger z.B. 1 für Call Control 1*/
       char sour;            /*Absender z.B. P für Prozess Manager*/
       messages requ;        /*Meldung z.B. aAkt*/
       int add1;             /*Adresse 1 z.B. 1 für Teilnehmer 1*/
       int add2;             /*Adresse 2 z.B. 2. für Teilnehmer 2*/
       tlnRole role;         /*Tln-Role (a,b,f)*/
       int time;             /*optional*/
};
```

Beispiel: Versenden einer Connect-Nachricht der Teilnehmer 30 und 31 von einer Call Control-Instanz an das Markierungsmodul:

```
header_K cHeaderC;            // Variable vom Typ header_K anlegen
cHeaderC.dest='M';            // Markierung als Empfänger
cHeaderC.sour='C';            // CallControl als Absender
cHeaderC.requ=con;            // Connect-Nachricht
cHeaderC.add1=0x30;           // Teilnehmernummer 30
cHeaderC.add2=0x31;           // Teilnehmernummer 31
//cHeaderC.role und cHeaderC.time werden für diese Nachricht nicht benötigt
PbxCC->SendK(&cHeaderC);      // Nachricht verschicken
```

Wie im Quellcode ersichtlich können nichtbenötigte Elemente der Struktur einfach weggelassen werden.

6.3 Der Prozessmanager

Der Prozessmanager dient der Vorverarbeitung und Verwaltung der Teilnehmeraktivitäten (z.B. Teilnehmer wird aktiv). Seine Aufgabe besteht darin, die Zustände der einzelnen Teilnehmer (A-Teilnehmer, B-Teilnehmer, Frei) zu verwalten, und Call Control-Instanzen zu erzeugen bzw. zu beenden. Wird ein Teilnehmer zum Beispiel aktiv und er ist in noch keinem Call involviert, muss der Prozessmanager diesen als A-Teilnehmer verwalten und eine neue Call Control-Instanz erzeugen.
Zur Speicherung von Teilnehmerinformationen steht die globale Struktur *„tlnTabelle“* zur Verfügung.

```
struct tlnTabelle{
       int inst;              /*CC-Instanz für Teilnehmer*/
       tlnRole role;          /*Teilnehmerrolle*/
};
```

Diese Struktur beinhaltet zum Einen die Integer-Variable *„inst“* welche die für den Teilnehmer aktuell zugewiesene Call Control-Instanz beinhaltet. Zur Vereinfachung der Programmierung kann hierfür die Rufnummer des A-Teilnehmers (also 1,2,3 oder 4) genutzt werden. Die zweite Variable vom Typ *tlnRole* kann die Werte a (A-Teilnehmer), b (B-Teilnehmer) oder f (Frei) annehmen. Da in diesem Versuch eine Call Control mit 4 Teilnehmern realisiert werden soll, muss im Prozessmanager ein Array vom Index 4, der Struktur *„tlnTabelle“*, deklariert werden. Die Deklaration des Arrays befindet sich bereits in der Datei „PbxManager.h“.

```
tlnTabelle tTab[4];
```

Für jeden Teilnehmer kann somit dessen zugewiesene Call Control und Role gespeichert werden. Ist z.B. der Teilnehmer 2 in keinem Call involviert, also auch keiner Call Control zugewiesen, würde in *„tTab“*

```
tTab[2].inst=0xff;    // keiner Call Control-Instanz zugewiesen
tTab[2].role=f ;      // Teilnehmer ist Frei, also weder A- noch B-Teilnehmer
```

stehen. Wurde hingegen Teilnehmer 2 von Teilnehmer 3 angerufen, würde in „tTab“

```
tTab[2].inst=3;       // Call Control-Instanz entspricht A-Teilnehmer
tTab[2].role=b;       // Teilnehmer 2 ist B-Teilnehmer (Teilnehmer 3 wäre A-Teilnehmer)
```

stehen.

Die Implementation der Prozessmanager-Logik erfolgt in der Datei „PbxManager.cpp“. Innerhalb dieser Datei steht die Methode *„PbxMessageManager(header_K headerM)“* zur Verfügung. Die übergebene Variable *„headerM“* vom Typ *„header_K“* enthält dabei die zu verarbeitende Nachricht. Die Implementation für den Empfang einer Wahlziffer und deren Weiterleitung an eine Call Control kann dabei wie folgt aussehen.

```
void __fastcall TPbxManager::PbxMessageManager(header_K headerM){
       x=headerM.add1;

       switch (headerM.requ) {
       case wz0:   // wzi
       case wz1:   // wzi
       case wz2:   // wzi
       case wz3:   // wzi
       case wz4:   // wzi
       case wz5:   // wzi
       case wz6:   // wzi
       case wz7:   // wzi
       case wz8:   // wzi
       case wz9:      // wzi
              if (tTab[x].role==a) {
                     mHeaderM[x].dest=tTab[x].inst;       // Ziel ist Call Control
                     mHeaderM[x].sour='P';                // Absender ist Prozess Manager
                     mHeaderM[x].requ=aWzi;               // A-Teilnehmer wählt Ziffer
                     mHeaderM[x].add1=x;                  // Nummer A-Teilnehmer
                     mHeaderM[x].add2=wzi[headerM.requ];  // gewählte Ziffer
                     PbxCC->SendK(&mHeaderM[x]);          // “Container” versenden
              }
              break;
       case akt: // Teilnehmer ist aktiv geworden
                     .
                     .
                     .
              break;
       case …
       }
}
```

6.4 Die Call Control

Die Call Control dient zur Steuerung der Ruftöne, Hörtöne und der Gespräche. Zur Speicherung der entsprechenden Zustände steht die Variable „cTab“ vom Typ der globalen Struktur *„callTabelle“* zur Verfügung.

```
struct callTabelle{
       callStates cState;           /*Zustand des Calls*/
       int aTln;                    /*Teilnehmerrolle*/
       int bTln;
};
```

Diese Struktur beinhaltet die Variable *„cState“* vom Typ *„callStates“*, welche zur Speicherung des Zustandes des Calls genutzt wird, sowie die Integer-Variablen *„aTln“* und *„bTln“*. „cState“ kann dabei die folgenden Werte annehmen:

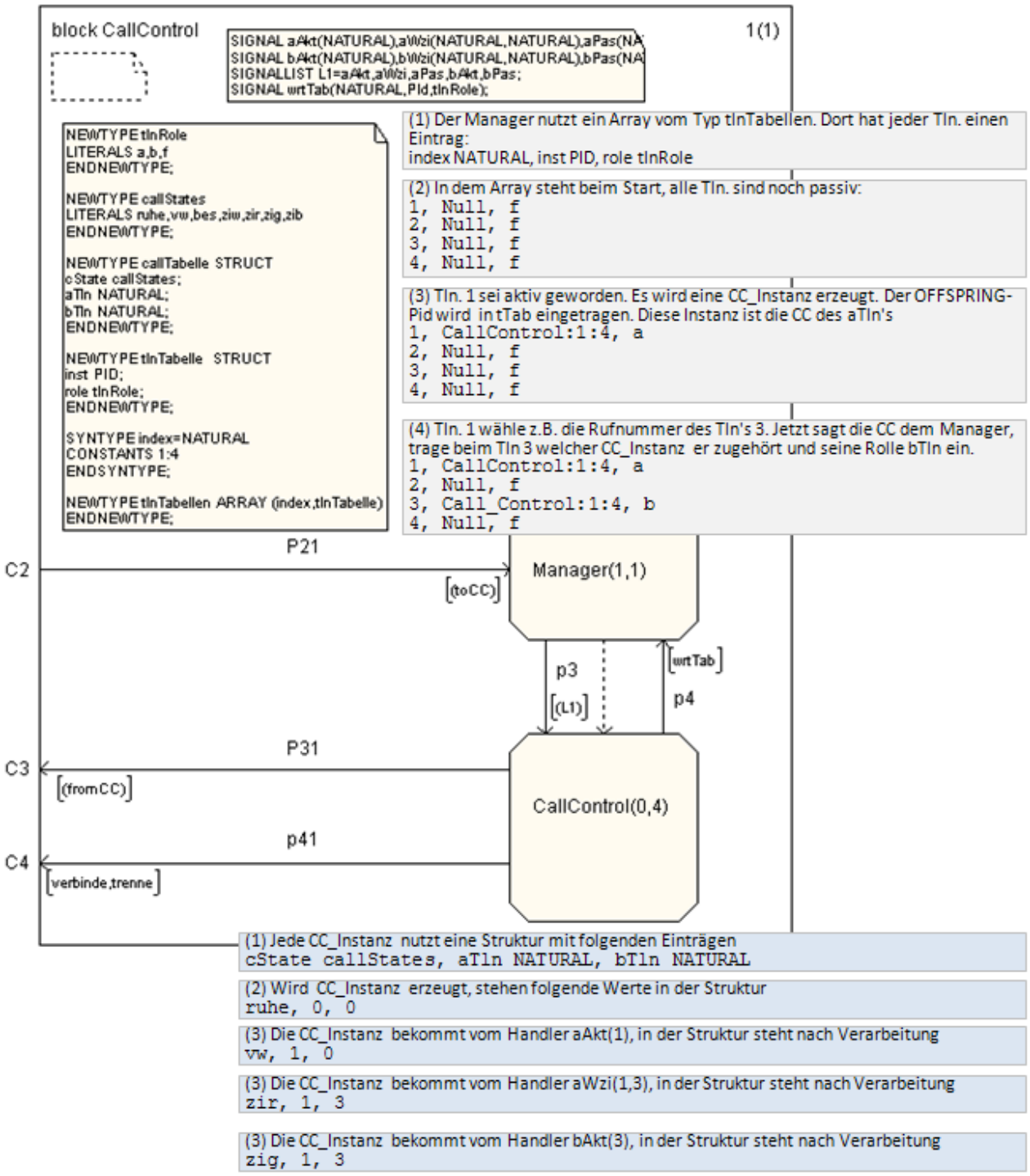
```
typedef enum {fre,vor,ziw,zir,zib,bes,zig,ebp,eap,ruhe,vw,waitofpas} callStates;.
```

Um Ruf- und Hörtöne bei Nichtaktivität des Nutzers zeitgesteuert abschalten zu können, ist es notwendig Timer in das Programm zu integrieren. Hierfür wurden bereits drei Timer „t_BZ“, „t_RZ“, „t_WZ“ in der Klasse „PbxCallControl“ implementiert. Zum Starten eines Timers ist folgender Quellcode notwendig (Beispiel für Timer tRZ).

```
t_RZ->Enabled=true;
```

Die Implementation der Call Control-Logik erfolgt in der Datei “PbxCallControl.cpp“. Innerhalb dieser Datei steht die Methode „*PbxMessageCallControl(header_K headerC)*“ zur Verfügung. Die übergebene Variable *„headerC“* vom Typ *„header_K“* enthält dabei die zu verarbeitende Nachricht. Die Verarbeitung einer eintreffenden „*aAkt*“-Nachricht (Teilnehmer A wurde aktiv) kann dabei wie folgt aussehen.

```
switch (cTab.cState) {
        case ruhe:                                          // Call Control-Zustand:ruhe
              switch (headerC.requ) {
              case aAkt:                                    // Empfangene Nachricht
                     cTab.cState=vw;                        // neuer Zustand der Call Control
                     x=headerC.add1;                        // Nummer des A-Teilnehmers
                     cTab.aTln=x;                           // A-Teilnehmer in Call-Tabelle
                     cHeaderC.dest='R';                     // Empfänger der Nachricht
                     cHeaderC.sour='C';                     // Absender der Nachricht
                     cHeaderC.requ=wz;                      // Wählzeichen ausgeben
                     cHeaderC.add1=cTab.aTln;               // A-Teilnehmer
                     cHeaderC.add2=0;                       // ungenutzt
                     PbxCC->SendK(&cHeaderC);               // “Container” versenden
                     t_WZ->Enabled=true;                    // WZ-Timer setzen
                     break;
              }
              break; // ruhe
        case vw:
              switch (headerC.requ) {
                     .
                     .
                     .
}
```


block CallControl
1(1)
SIGNAL aAkt(NATURAL),aWzi(NATURAL,NATURAL),aPas(NA
SIGNAL bAkt(NATURAL),bWzi(NATURAL,NATURAL),bPas(NA
SIGNALLIST L1=aAkt,aWzi,aPas,bAkt,bPas;
SIGNAL wrtTab(NATURAL,Pid,tlnRole);
NEWTYPE tlnRole
LITERALS a,b,f
ENDNEWTYPE;
NEWTYPE callStates
LITERALS ruhe,vw,bes,ziw,zir,zig,zib
ENDNEWTYPE;
NEWTYPE callTabelle STRUCT
cState callStates;
aTln NATURAL;
bTln NATURAL;
ENDNEWTYPE;
NEWTYPE tlnTabelle STRUCT
inst PID;
role tlnRole;
ENDNEWTYPE;
SYNTYPE index=NATURAL
CONSTANTS 1:4
ENDSYNTYPE;
NEWTYPE tlnTabellen ARRAY (index,tlnTabelle)
ENDNEWTYPE;
(1) Der Manager nutzt ein Array vom Typ tlnTabellen. Dort hat jeder Tln. einen Eintrag:
index NATURAL, inst PID, role tlnRole
(2) In dem Array steht beim Start, alle Tln. sind noch passiv:
1, Null, f
2, Null, f
3, Null, f
4, Null, f
(3) Tln. 1 sei aktiv geworden. Es wird eine CC_Instanz erzeugt. Der OFFSPRING-Pid wird in tTab eingetragen. Diese Instanz ist die CC des aTln's
1, CallControl:1:4, a
2, Null, f
3, Null, f
4, Null, f
(4) Tln. 1 wähle z.B. die Rufnummer des Tln's 3. Jetzt sagt die CC dem Manager, trage beim Tln 3 welcher CC_Instanz er zugehört und seine Rolle bTln ein.
1, CallControl:1:4, a
2, Null, f
3, Call_Control:1:4, b
4, Null, f
C2
P21
(toCC)
Manager(1,1)
p3
(L1)
wrtTab
p4
C3
P31
(fromCC)
CallControl(0,4)
C4
p41
verbinde,trenne
(1) Jede CC_Instanz nutzt eine Struktur mit folgenden Einträgen
cState callStates, aTln NATURAL, bTln NATURAL
(2) Wird CC_Instanz erzeugt, stehen folgende Werte in der Struktur
ruhe, 0, 0
(3) Die CC_Instanz bekommt vom Handler aAkt(1), in der Struktur steht nach Verarbeitung
vw, 1, 0
(3) Die CC_Instanz bekommt vom Handler aWzi(1,3), in der Struktur steht nach Verarbeitung
zir, 1, 3
(3) Die CC_Instanz bekommt vom Handler bAkt(3), in der Struktur steht nach Verarbeitung
zig, 1, 3

Anlagen

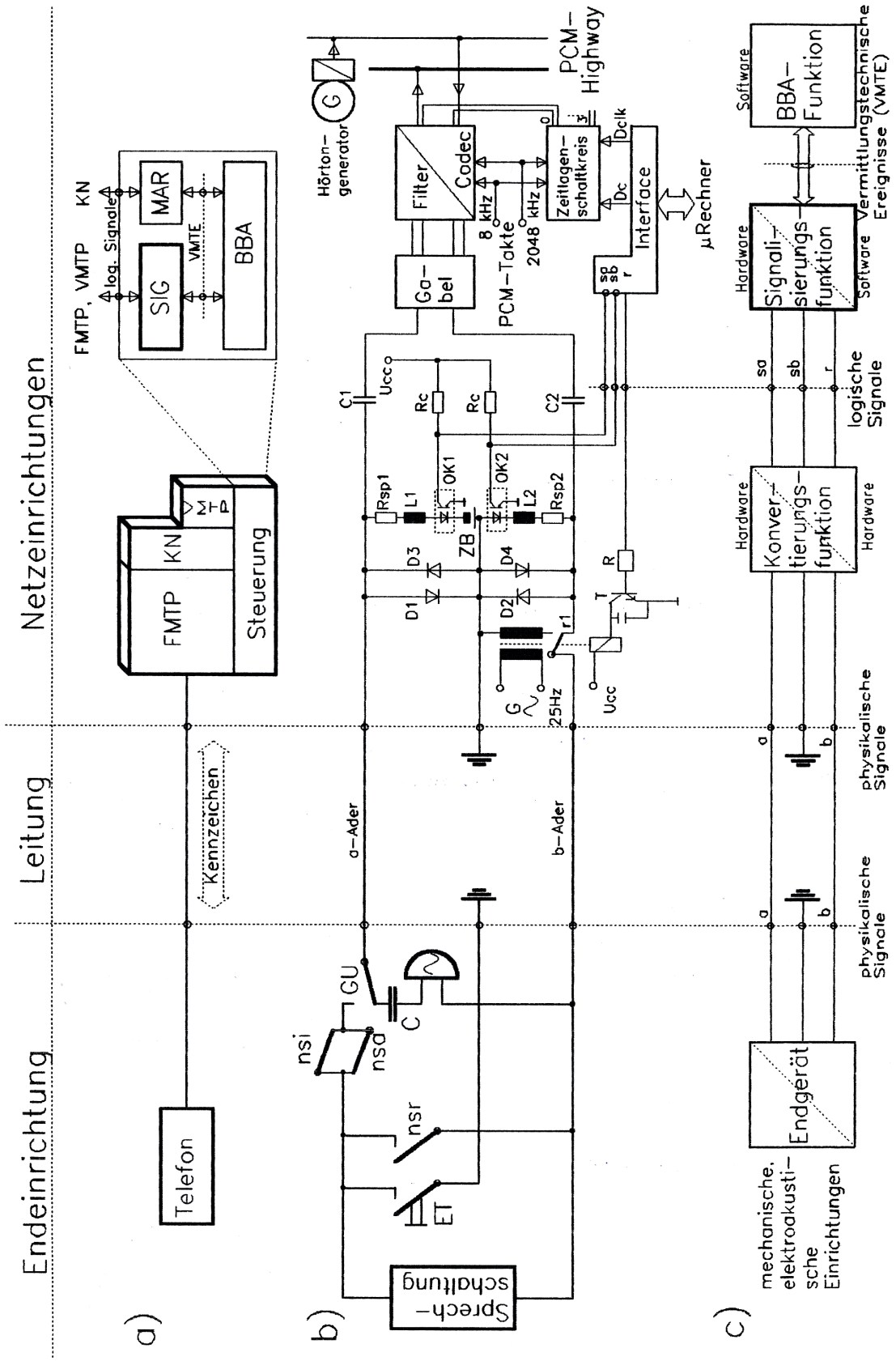


Abbildung 14: System Endgerät/Vermittlung

Literaturverzeichnis

/1/ W. Cimander; H. Stürz: Automaten, Berlin 1974

/2/ S. Wendt: Entwurf komplexer Schaltwerke, Springer-Verlag 1974

/3/ L. Winkler, K. Kissing: Ein Beitrag zum Entwurf und zur Realisierung digitaler Vermittlungseinrichtungen, insbesondere Nebenstellanlagen, IH Mittweida 1982
: