digisolutions

SC5BOS

Scheduling C5 Basic Operating System

Version 0.9e

Ein Projekt von

DB1OTM, Thomas Müller

DO1FJN, Jan Alte

Dokumentation für Programmierer

Inhalt

Inhalt

Terminologie im Dokument

- Bootloader Teil des SC5BOS der mit dem Benutzer interagiert.
- C5 Bezeichnet das Telefon Siemens C5 (auch ABB C45-5)
- SC5BOS Sheduling C5 Basic Operation System, das Betriebssystem welches hier vorgestellt wird
- SC5PRG Sheduling C5 Program, ein x86-Program das im einem C5 mit SC5BOS funktioniert.

Einleitung

Inhalt dieser Dokumentation

Dies Dokumentation umfaßt die technische Beschreibung des Siemens C5 und des Betriebssystems SC5BOS. Diese Beschreibung ermöglicht einen Einstieg in die Erstellung eigener Programme.

Ziele

Bei diesem Projekt ging es vorrangig darum, kostengünstig und mit minimalem Aufwand das Siemens C5 Telefon in einen FM-Amateurfunktranceiver für das 70cm Band (430-440 MHz) zu verwandeln. Begonnen hat alles mit der sehr detaillierten Anleitung von DL6INT (Uwe Henning). Diese findet man im Internet unter:

http://home.t-online.de/home/dl6int-1/c5/c5idx.htm

Leider erforderte der Umbau des C5 nach dieser Anleitung einigen Aufwand in der Realisierung und hat gegenüber dieser „Softwaremethode“ viele Nachteile. Da der Autor Informatik studiert hat, und sich Hobbymäßig mit Amateurfunk beschäftigt, kam er auf die Idee ein kleines Betriebssystem für das C5 zu entwickeln und kostenfrei im Internet bereitzustellen. Dieses Vorhaben wurde inspiriert durch die Veröffentlichung einiger Details des C5-Software-Innenlebens durch DB1OTM (Thomas Müller). Auf seiner Internet-Seite findet man die Anfänge der NEC-Controller-Programmierung.

http://www.qsl.net/db1otm/c5.html

Es entstand eine Zusammenarbeit, die auf einen gegenseitigen Informationsaustausch beruhte. Basis der Entwicklung ist übrigens die von DB1OTM disassemblierte Originalsoftware, die einige Fragen beantwortet, jedoch 10mal so viele Neue aufwirft. Beispiele dafür ist die C-Netz-Signalisierung. Es ist dem Autor noch nicht gelungen, die Funktionsweise des im EP200 integrierten FSK-Demodulators und der Art der Modulation (DA-Wandler oder DSP) zu ergründen.
Das Ziel des Autors besteht nun darin, dieses Betriebssystem und seine „Brüder“ weiterzuentwickeln, Fehler zu eliminieren und den Funktionsumfang des Applicationinterfaces (API) zu erweitern. Es ist dabei nicht Aufgabe des SC5BOS telefon- oder funkbetriebsspezifische Funktionen bereitzustellen, sondern nur die Schnittstelle um einfach mit dem Benutzer des Gerätes zu interagieren (Ansteuerung des Hörers, PC-Vernindung).
Da sich bisher leider niemand (nach dem Kenntnisstand des Autors am 1.1.2003) weiter der Programmierung für dieses Telefon beschäftigt, bietet er auch eine einfache Steuerungssoftware „C5-SUP“ (Simple User Program) an, die das Telefon in einen FM-Tranceiver verwandelt.

Beschreibung des Projektes „SC5BOS"

Das Projekt „SC5BOS" beschäftigt sich mit der Softwareentwicklung für das alte C-Netztelefon Siemens C5. Parallel werden SC3BOS, SC4BOS und PP5BOS „softwarekompatibel" mitentwickelt (für die Telefone Siemens C3, C4 und Philips Porty FG51 bis FG53). Alle Telefone verfügen über einen NEC V25 oder V40 Microcontroller (µC). Der NEC V25 / V40 ist softwarekompatibel zu einem i80186 und besitzt zusätzlich noch erweiterte Befehle. Er ist ansonsten ein klassischer Controller (keine CPU) mit UART, Ports, DMA etc.). Software kann deshalb mit „alten" PC-Assemblern und Compilern erstellt werden. Das C3 und das C5 haben neben dem µC noch einen DSP 56001 von Motorola onboard. Dieser ist über das Host-Interface und der „glue logic" (EP200 im C5) mit dem µC verbunden.
Der Autor hat die hier vorgestellte Software mit dem A86-Assembler und dem jloc-Linker erstellt. Der A86-Assembler kennt ein paar der zusätzlichen NEC-Befehle (leider nicht alle) und verfügt über eine einfaches Segmentierungs-System. Der jloc-Linker wertet die von A86 erstellte OBJ-Datei aus und erzeugt mit Hilfe einer Kontrolldatei ein binäres Image des Programmes. Als „Entwicklungsumgebung" diente der Ultraedit (http://www.ultraedit.com/), der Syntaxhighlighting und das Einbinden von Programmen (hier z.B. make) beherrscht. Leider ist dieses Programm Shareware, dessen Registrierung mit $35 noch human ausfällt.
Neben der Softwareentwicklung konnte sich der Autor auch nicht um Hardwareentwicklung drücken. Um das C5 mit dem PC zu Verbinden, wurde ein serieller Pegelwandler (TTL ß à RS232) benötigt. Als einfache Lösung wurde ein Max202 und ein kleiner 5V-Stabi auf einer Lochrasterplatine zusammen mit 2 Pfostensteckern gelötet. Mittlerweile existiert eine winzige Platine, die mit in ein Sub-D9-Gehäuse gesetzt wird. Die Beschaltung entspricht der im Max202-Datenblatt. Angeschlossen wird diese Baugruppe an die 26pol. Buchse des C5 (HDB26 benötigt). Über diese Buchse erfolgt auch gleich die externe Stromversorgung. Folgende Pins müssen mit dem Max202 verbunden werden: V24-TXD (Pin13) und V24-RXD (Pin20).
Neben dem Seriellen Anschluss ist leider auch ein Programmiergerät erforderlich gewesen, um einen leeren Flash-Baustein mit der Software zu beschreiben. Erst mit dem programmierten Flash konnte erstmalig „eigene" Software ins C5 gelangen. Als Flash verwendet der Autor den PLCC32-Typ 29F040 der Firmen AMD, ST, TI und Hyundai. Alternativ einsetzbar sind auch die Typen 29F002TC / BC. In jedem Fall muss das Original-ROM aus dem C5 entfernt und eine PLCC32-Fassung eingelötet werden (wenn nicht schon vorhanden).

Die Software SC5BOS und C5-SUP sind nicht das einige was im Laufe der Entwicklungsarbeit entstand. Es entstand eine ganze Reihe von Testprogrammen, die im C5 ausführbar sind. Dazu zählen u.a. FlashWRT, DSPTest, C5Demo, I2Ctest und C5Test. Da die Programme ihre (Test-) Aufgabe erfüllt haben werden sie nicht mehr weiterentwickelt. Viele Routinen für SC5BOS entstanden durch diese Programme. Eine Sonderstellung nimmt dabei C5-Demo ein. Wie am Name erkennbar ist es eine einfache Demonstrationssoftware, die die Kernfunktionen für einen FM-Funkbetrieb implementiert. C5-Demo ist wie C5-SUP in Assembler geschrieben und wird samt Quelltext für Entwickler bereitgestellt.

Einige Features von SC5BOS in Kurzform:

- sichere Kommunikation mit dem PC über ein paketorientiertes Protokoll (PCP)
- neue Software wird einfach bei geschlossenem Gerät ins C5 übertragen
- Software kann für Tests im Ram (30Kbyte verfügbar) ausgeführt und für den regulären Betrieb in einen Sektor des Flashrom geschrieben werden.
- SC5BOS ist updatefähig – neuere Versionen werden ins Ram geladen und schreiben sich ins Flashrom.
- mehrere Funktionen können durch Ereignisroutinen („Handler") parallel laufen.
- Digitale Betriebsarten wie PacketRadio können ohne Modem realisiert werden, da der DSP Modulation und Demodulation übernehmen kann (im Bereich von 240-3200Hz). 1k2-AFSK z.B. ist daher möglich.
- Entwickler brauchen sich nur noch um die eigentliche Aufgabe des Programms kümmern, den Rest erledigt dann SC5BOS (I/O, Tastenabfrage etc.)

Unweigerlich entstand auch PC-Software: Diese unter Delphi5 geschriebenen Tools sind für die Kommunikation mit dem Telefon (oder „kompatiblen" Geräten) gedacht. Kern der Kommunikation ist das Übertragungsprotokoll PCP (Packet Communication Protocol. Es spezifiziert eine auf Paketen aufgebaute serielle Kommunikation zwischen 2 Geräten (hier C5 und PC). Die Fähigkeiten der Windows-Software in Kurzform:

- OLE-Server für die Kommunikation (PCP) über eine serielle (Standart-) Schnittstelle-
- Einsehen vom Ram- und Rominhalten per PCPMemoryViewer
- Ausgabe von Textmeldungen des Telefons per PCPMessageViewer
- Übertragen, Ausführen und „flashen" von Programmen durch den „Telefonprogrammierer"
- Steuern von C5-SUP durch das „SUP - Control Program" (noch in Entwicklung).

(Diese Software wird in einem extra Dokument behandelt.)

Lastenheft SC5BOS

Im Lastenheft sind die implementierten Features von SC5BOS zusammengefasst. Es diente bei der Entwicklung als „Checklist“ welche Fähigkeiten noch nicht realisiert sind. Mittlerweile ist das Lastenheft komplett erfüllt, die Entwicklung findet jetzt als Erweiterung und Verbesserung dieser Punkte statt.

- SC5BOS soll in einem 16Kbyte großen geschützten Sektor des Flashroms Platz finden.
- Initialisierung der Telefon-Hardware (definierter Betriebszustand):
  - Prozessorregister
  - Interrupt-Vektortabelle
  - weitere programmierbare Bausteine (EP200, DSP)
  - Telefonhörer
  - externe serielle Schnittstelle (57600baud, 8N1)
- Implementierung eines einfachen Kommunikationsprotokolls das eine sichere Datenübertragung gewährleistet. Daher ist eine Prüfsumme (CRC 8bit oder CRC 16bit nach CCITT) Bestandteil jedes Paketes.
- Der Telefonhörer soll bereits Meldungen ausgeben können.
- Im Flash wird nach dem Start das jüngste Programm (höächste Sektornummer) gesucht
- Ohne Programm startet ein Bootloader. Es stellt Programmier- und Testfunktionen bereit. Darunter fallen:
  - Ausgabe Version, Erstellungsdatum, Flashgröße und -kennung
  - Programmstart
  - CRC16 des Flashbausteins
  - Flashsektor oder gesamten Flash löschen (bis auf SC5BOS)
  - Test der Seriellen Schnittstelle 0 (für PC-Kommunikation wichtig)
  - Einstellen der Übertragungsparameter (Baudrate), abweichend vom Standart (z.B. 9600baud).
  - Speichertest
- Der Funktionsumfang des API‘s ist einfach zu erweitern
- SC5BOS kann durch den PC gesteuert werden:
  - Laden und Ausführen eines Programm im Speicher (Ram).
  - Schreiben von Daten und Programmen in den Flashrom („Programmiermodus“).
  - Abschalten und Neustart des Telefons.
  - Auslesen von Ram- und Flashrominhalten.
  - Abfrage des Betriebzustandes und der Versionsnummer .
- Über ein API werden Basisfunktionen den Anwendungsprogrammen zugänglich gemacht.
- Anwendungsprogramme können Ereignisbehandlungsroutinen („Handler“) installieren die einer Standart FAR-Funktion entsprechen.
- Es können 16 verschiedene Handler benutzt werden. Die Daten werden in Prozessorregistern übergeben.
- ...

Ideensammlung für Kreative

Hardware

- Statt Hörer Anschluß von Bedienteilen an den BF-Bus.
    - Display 4x20 Zeichen oder Grafikpanel (320x240 dots)
    - Impulsdrehgeber für Volume – Frequenz (Umschaltung durch Drücken, Timeout)
    - 4 Softkeys am Display-Rand
    - Mikrofonanschluss für Handmikrofone
    - kleiner Lautsprecher
    - Einbaurahmen für den KFZ-Betrieb
- Ram-Erweiterung (maximal +448kByte möglich).
- SmartMediaCard (SMC) Adapterplatine (3.3V-Treiber, Kontaktiereinrichtung)
- Benutzung der „Office"-Platine (TAE-Anschluss) für FAX Senden/Empfang
- Infrarot-Fernbedienung
- Zusatzplatine (wie das TAE-Modul) mit Zusatzcontroller und BF-Bus-Anschluß (z.B. PIC-Modem, TNC, APRS oder Steuerung)
- Kopplung an ein anderes Funkgerätes oder Telefons (programmierbarer Umsetzer).
- Cinch-Ein- und -ausgänge zum Durchschleifen von AutoRadio-NF
    - Analoger NF-Mischer

Software

- Erzeugen und Verarbeiten von
    - CTCSS (nur per DA-Wandler und dem NEC), DTMF
- PacketRadio Modem-Funktion für 1k2 / 9k6 (Handware) oder 5,28k (C-Netz) bzw. 4k8 (Afu)
- SSTV-Decodierung (Speichern des Bildes in RAM-Erweiterung oder einer SMC)
- Benutzung der C-Netz-Karten (Telefonbuch-Speicher, in Vorbereitung)
- Einsatz von Prozessorkarten (Taktfrequenz: 4,9152MHz) zum Speichern größerer Datenmengen
- Ablage von Betriebsparametern auf Chipkarten
- Benutzung der C-Netz-Rufnummer (da eindeutig) für ein Signalisierungssystem
- Mischbetrieb von PacketRadio und Sprache
- Verteilte Signalisierung: Nutzung der Zeitkompression und der Datenübertragung in den Zeitschlitzen (ca. 320bit/s)
- Klingelfunktion
    - Über ein PR-Paket (UI-Frame)
    - Funkruf
    - Über DTMF-Folge („Telefonnummer")
- Funkruf (Pager) Empfangsfunktion, Skyper-Emulation
- DSP-Sprachverbesserungsfähigkeiten
    - Rauschunterdrückung, -Sperre á la FM-Select
    - Bandpassfilter, Notchfilter
- Aushandeln von Sendeleistungen mit Simplexbetrieb per verteilter Signalisierung.
- Aussenden von APRS (Serieller Eingang für „GPS-Maus")
- HDS (HamDataSystem) – wie RDS über die verteilte Signalisierung. Aussenden von Rufzeichen, Locator, Infotext etc.
- Einbau eines 2m MiniEmpfänger (aus Funkamateur 9/01 S.999)
    - ohne ATMEL und NF-PA, dafür digitales Poti am AF-Codec.
- DTMF-Gesteuerter Anrufbeantworter -> Über ext. Speichermedium (SMC, Chipkarte, extra-Ram)
- Fernabfrage Anrufbeantworter über anderes Funkgerät
- Bei Funkgerätekopplung als Relais konfigurierbar, Fernbedienung über DTMF (Ausgabe QSY)
- CW-Dekodierung und Darstellung im Display (Kennungen von Relais)
- Über Ser. Schnittstelle Nordlinks TF-Emulation mit erweitertem Spezialbefehlssatz (@C5 ...) als Standard, 6Pack wählbar)
- Relais und DigipeaterListe (Updatebar) von EU/DL
  à durch Eingabe das Locators sind 70cm Relais auswählbar (im Umkreis erreichbare)

Umbau der digitalen Hardware

Leider kommt man beim Einsatz eines Flash-Bausteins nicht um das „Strippenziehen' und Patchen herum. Nachfolgend befindet sich eine kleine Einbauanleitung für ein 29F010, 29F002 oder 29F040-Flash.

1. Vorsichtiges Auslöten des D810 (ROM), wenn dieser bestückt war (Achtung bitte keine Pats abreisen oder Leiterbahnen durchtrennen! **à** siehe Tip)
2. **Nur bei 29F040 erforderlich*:** Auftrennen der Verbindung von Pin1 (A18) und Pin32(Vcc). Diese befindet sich auf der Oberseite direkt an den PLCC-Pads. Achtung: Pin32 bleibt weiter an Vcc
3. **Nur bei 29F040 erforderlich*:** Verbinden des PLCC-Pin1 mit der A18 des µC (Pin8) durch einen Fädler.
4. Entfernen des Pull-Up-Widerstands R885 des WE (Pin31). Dieser befindet sich auf der Platinenunterseite unter dem PLCC-Pads (hier natürlich nicht sichtbar)
5. Wenn nicht bestückt: Einsatz eines Stütz-Kondensators (C805, ca. 100n) unterhalb des R885.
6. Verbinden des WE (Pin31 des PLCC-Pads) mit dem WE am SRAM (Pin27). Diese Verbindung kann elegant auf der Unterseite von den Durchkontaktierungen aus verlegt werden.
7. Auflöten einer PLCC32-Fassung. Achtung bei dem Fädler am Pin1. Tip: Durch die Entfernung des Fassungsbodens können die Pins mit einem üblichen Feinlötkolben angelötet werden.
8. Fassungsboden wieder hineinlegen (fixieren).
9. Programmiertes Flash einsetzen.

***)** Diese Schritte können auch bei Einsatz eines kleineren Flashroms (29F010, 29F002) gemacht werden. Dann kann später Dieser durch einen 29F040 ersetzt werden.
Auch wenn ein Original-EProm (27C101) eingesetzt werden soll, behindert der hier beschriebene Umbau nicht die Funktion.

Tip:
Um ein eingelöteten ROM sauber zu entfernen kann man auch wie folgt vorgehen: Man nehme ein sehr scharfes Messer (Teppichcutter z.B.) und setzt dieses an einem Beinchen des IC's direkt am Gehäuse an. Jetzt drückt man vorsichtig senkrecht nach unten und voilá: das Beinchen ist durchtrennt. Wenn man so alle Beinchen abgedrückt hat, fällt nimmt man den IC heraus und lötet mit Entlötlitze die Beinchen ab. Bei dieser Methode ist es wichtig, das Messer direkt am IC anzusetzen und nur zu drücken. Bewegt man das Messer hin- und her ist die Gefahr gegeben darunter liegende Leiterbahnen durchzuritzen.

Umbau der analogen Hardware

Dieser Umbau erfolgt weitestgehend nach DL6INT und berücksichtig die dort gesammelten Erfahrungen. Eine angepaßte Version befindet sich auf der Web-Seite des Autors (http://www.digisolutions.de/).
Hier befinden sich nur Stichpunkte der Vollständigkeit halber:

- Oszillator der PLL verstimmen (ca. 30MHz tiefer, breitbandiger)
- Austausch der Sendefilter (2 SAW-Filter) gegen einen oder zwei Helix-Filter.
- Austausch von 3 weiteren Kondensatoren (Treiber und AUDI-Frequenzgang)
- Spannungsteiler für optionale (Versorgungs-) Spannungsmessung
- Einbau einer Antennenankopplung z.B. einfache Diodenumschaltung unter Verwendung des Duplexfilters als Bandpaß oder Ersatz des Duplexfilters durch eine Platine (z.B. von DL8SDL)
- Einbau einer optionalen Platine für eine Packet-Radio-Ankopplung (Analogschalter)

Konzeptdokumentation SC5BOS

Das Grundkonzept der Programmierung des Telefons ist die Unterteilung der „Firmware“ in Betriebssystem und Anwendungsprogramm. Nur durch diese Zweiteilung ist eine einfache Programmentwicklung für Dritte denkbar. Da die Funktionen im Betriebssystem optimiert in Assembler programmiert sind reichen dem Betriebssystem 16Kbyte Speicherplatz. Durch die fast identischen API’s der verschiedenen BOSs sind die Programme zudem einfach portierbar, wie der Autor am C5-SUP -> C3-SUP und PP-SUP bereits demonstriert hat. Da die Programme die Funktionen des BOS nutzen, kann ihre Implementierung auch in einer Hochsprache erfolgen, ohne das die Ausführungsgeschwindigkeit spürbar abnimmt. Auch bleibt die Dateigröße klein und der Quelltext übersichtlich. Man erspart sich teilweise auch das umständliche Ansprechen der Hardware, welches teilweise nur in Assembler realisierbar ist (z.B. Stackinitialisierung, Interruptroutinen etc.). Auch braucht sich der Programmierer keine Gedanken mehr um die richtige Segmentierung machen. Z.Z. können Programme als Ein- oder Zweisegment (Tiny, Small) erstellt werden. Eine direkte Umwandlung von COM- oder EXE-Formaten ist noch in Vorbereitung, so das man z.Z. die OBJ-Datei mit jloc linken muß.

Das SC5BOS ist in dem Flash in einem kleinen Block am oberen Aderessende untergebracht. Die für den Einsatz im C5 in Frage kommenden Flashtypen (z. B. 29F002TC, 29F040) besitzen einen 16Kbyte-Sektor (29F002TC) oder einen 64Kbyte-Sektor (29F040) am oberen Aderessende. Da Sektoren in einem externen Programmiergerät komplett gegen Überschreiben geschützt werden können, kann ein solch geschütztes SC5BOS-Flash auf fest ins C5 eingelötet werden. Da jedoch wichtige Aufgaben wie die komplette serielle Kommunikation vom SC5-API gekapselt werden (müssen), ist jedoch erst einmal ein gesockelter Flash mit ungeschützten SC5BOS vorzusehen. Bei Bedarf kann hier das gesamte Inhalt neugeschrieben werden, was jedoch kritische Momente erfordert (SC5BOS einen Moment lang nur noch im SRAM)!

Betriebssystemerfordernisse:

- Einfaches Multitasking über den Timer2 (10ms-Systick)
- Application Interface (API) über Softints (ähnlich dem MSDOS) mit Parameterübergabe in den Registern
- integrierter Bootloader für wichtige Funktionen (Tests, Informationsausgabe)
- Funktionen:
    - Test, ob Benutzerprogramm (SC5PRG) im Flash geladen ist
    - Kopiert bei Bedarf SC5BOS vom Flash ins SRAM und führt es von dort aus
    - PCP: Update SC5BOS über externe serielle Schnittstelle
    - PCP: Neue Software / Update -> Flash paketweise neu beschreiben
    - PCP: Programme in SRAM laden und ausführen
    - PCP: Aktivierung bei laufender Anwendersoftware über die serielle Schnittstelle möglich.
- Unterteilung
    - Bootloader (BL) als Bestandteil des SC5OS
    - Application Interface (API) als Bestandteil des SC5BOS
    - Signalisierung über Soft-Interrupts (Handler) durch das SC5BOS
    - C5-Programm (SC5PRG)
    - Texte und Funktionen, Menü, Ser. IO etc.

Bedienungsanleitung

Die Bedienungsanleitung ist nicht mehr Bestandteil dieser Dokumentation und wird extra bereitgestellt.
(SC5BOSManualv0.9e.pdf)

technische Beschreibung des SC5BOS

Funktionen und Fähigkeiten des SC5BOS und des Bootloaders

Nach dem Drücken des Einschaltknopfes am C5 springt der Prozessor den Reset-Vektor (0FFFF0h) an. An ihm befindet sich ein Sprungbefehl, der auf die Reset-Routine zeigt. In dieser Routine werden alle wichtigen Speicherbereiche und benötigte Peripherie initialisiert und eine Startmeldung (Licht, 3xBeep) zum Hörer geschickt. Danach sucht das Programm an den Segmentanfängen im eingesetzten Flash oder EProm nach der Kennung „SC5PRG“. Die Flash-Größe wird über eine Datenspiegelung detektiert (Daten spiegeln sich beim 128K und 256K-Typ). Findet SC5BOS eine Kennung, so startet es das Programm, das dort abgelegt ist. Ansonsten wird der Bootloader gestartet, das dem Benutzer ein paar Funktionen anbietet.
Durch die Initialisierung werden auch Behandlungsroutinen aktiviert, die z.B. den 10Hz-Watchdog triggern, die Pakete (Tastenbetätigungen) vom BF-Bus und dem Hörer entgegennehmen und weiterleiten, die On/Off-Taste überwachen, Tastenwiederholungen generieren, Pakete auf dem BF-Bus und der seriellen Schnittstelle ausgeben, PC-Anfragen / Befehle (Port0) beantworten und noch etliche Kleinigkeiten mehr erledigen.

Kommunikationsprotokoll

Für eine geordnete Kommunikation über die externe serielle Schnittstelle ist das SC5BOS verantwortlich. Es ist ein Kommunikationsprotokoll implementiert, das auf Paket-Transfer basiert. Das Protokoll stellt 10 Ports (eigene Kanäle auf der seriellen Leitung) zu Verfügung, von denen 2 vom SC5BOS benutzt werden. Die Kommunikation mit SC5BOS erfolgt der Einfachheit halber im PingPong-Betrieb. 5 Ports sind so genannte well-known-ports, deren Funktion bekannt und hier dokumentiert ist.

well-known-ports

Port	Implementiert in	Funktion(en)	Daten
0	SC5BOS	- Speicher (SRAM) auslesen - Speicher beschreiben - Programm starten - Programm unterbrechen - Gerät resetten, ausschalten - interne Testroutinen starten	Steuerpakete der Form Cmd, {Param}, Datenpakete (spez. Aufbau) (Master-Slave)
1	Debugging-Modul*	- Prozessorstatus ausgeben	Registerbank-Struktur
2	Bootloader, Anwenderprogramm	- Textmeldungen ausgeben	ASCII-Zeichenketten ohne Steuerzeichen
3	Anwenderprogramm	- Gerätesteuerung	Steuerpakete der Form Cmd, {Param} (Master-Slave)
4	Anwenderprogramm	- Packet-Radio 1k2	gekapselte AX25-Pakete
5	Anwenderprogramm	- Packet-Radio 9k6	gekapselte AX25-Pakete

*) Bisher nicht realisiert.

Die Steuerung von SC5BOS über Port0

Auf diesem Port werden Daten als Master–Slave-Kommunikation ausgetauscht. Das Telefon befindet sich in der Rolle eines Slaves. Der Master (PC) sendet dem Slave (Telefon) die hier definierten Kommandos und erhält darauf hin eine Antwort. Nach dem Absenden eines Requests muß ein TimeOut eingehalten werden, bevor der Request als Mißerfolg (Fehler in der Übertragung) gewertet wird. Ist dem Telefon ein Kommando unbekannt, so sendet es ein NAK-Paket zum Master.

Eine Übersicht der Kommandos:

Kommando	Code	Parameter	Response-Format / Inhalt
lese Gerätebezeichnung	00h	keine	ASCII-String (ohne Längenbyte)
lese Seriennummer	01h	keine	8-stelliger ASCII-String (nur Ziffern)
lese Softwareversion	02h	keine	Format: „vx.xxc“ (6Stellen)
lese Gerätestatus	03h	keine	2 Byte Flags + variable Anz. Bytes
lese Speicheraufbau	04h	keine	siehe Text
lese Speicherblock aus	10h	Typ, Anfangsadresse, Blockgröße	Speicherblock (nur Binärdaten)
gehe in Programmiermodus	11h	keine	ACK oder NAK
schreibe Block in Speicher	12h	Typ, Anfangsadresse, Binärdaten	ACK oder NAK
lese Flash-Hersteller / DeviceID	13h	keine	Word mit Hersteller- und DeviceID
lese Flash-Programmzyklusdauer	14h	keine	Byte (Dauer in µs)
formatiere Flash*	16h	keine	ACK oder NAK
lösche einzelnes Flashseg. *	17h	Segmentnummer (Flashherst. abh.)	ACK oder NAK
lösche Flash komplett*	18h	keine	ACK oder NAK
Updatesequenz einleiten2)	21h	keine	ACK oder NAK
Testsequenz ausführen	8xh	keine, im Telefon nicht benutzt	-
Programm ausführen	F0h	Einsprungadresse	ACK oder NAK
Programm stoppen	F1h	keine	ACK oder NAK
Programm abbrechen	F2h	keine	ACK oder NAK
Telefon abschalten	FEh	keine	ACK (unmittelbar davor) oder NAK
Soft-Reboot	FFh	keine	ACK (vor Reboot) oder NAK

* diese Befehle sind erst nach den Wechsel in den Programmiermodus möglich (sonst NAK-Response).
2) antwortet das µC-Gerät mit einem ACK, so können keine weiteren Befehle mehr behandelt werden, bis der Update vollständig ist (je nach Gerät einige Sekunden)

Die Kommandos im Einzelnen

(00h) „lese Gerätebezeichnung“
Das Telefon antwortet auf dieses Kommando mit einer Zeichenkette die eine Bezeichnung im Klartext darstellt. Die Zeichenkette enthält kein Längen- oder Ende-Zeichen mehr (die Länge wird durch das Längenbyte des Long Frame festgelegt). Dieses Kommando **muß jedes Gerät** unterstützen, da mit dessen Hilfe die Verbindung initialisiert wird (Test, ob ein Gerät antwortet).
Response-Beispiel:

„Siemens C5 with SC5BOS“

(01h) “lese Seriennummer”
Das Telefon antwortet auf dieses Kommando mit einer Zeichenkette die aus 8 Ziffern besteht.
Response-Beispiel:

„02455199“

(02h) “lese Softwareversion”
Das Telefon antwortet auf dieses Kommando mit einer formatierten Zeichenkette. Als Versionskennung wird als erstes Zeichen ein kleines „v“ übertragen, danach die erste Versionsziffer. Als Trennung folgt anschließend ein Punkt und 1 Subversionsziffer. Das letzte Zeichen ist das Release (kleiner Buchstabe von „a“ bis „z“ oder ein Leerzeichen) gefolgt von einem Leerzeichen. Die Anzahl der Nutzdaten ist somit auf 6 festgelegt.
Response-Beispiel:

„v0.9e “

(03h) „lese Gerätestatus“
Die Response auf dieses Kommando gibt dem Master eine Übersicht, welche Fähigkeiten das Telefon besitzt und in welchem Modus sich SC5BOS befindet. Die Response enthält als erstes Byte die CapFlags (Fähigkeitenanzeiger). Das 2. Byte ist z.Z. reserviert. Der Aufbau der CapFlags ist unten dargestellt. Evtl. folgende Bytes sind optional und geräteabhängig zu interpretieren.

CapFlags:

Bit	Bezeichnung	Beschreibung
0	Programmiermode vorhanden	immer „1“ bei SC5BOS
1	Update-Fähigkeit	immer „1“ bei SC5BOS (updatefahig)
2	Programmiermodewechsel	„1“, wenn jetzt ein Moduswechsel gerade möglich ist oder der Programmiermode bereits eingeschaltet wurde.
3	Updatewechsel	„1“, wenn der Update jetzt gestartet werden kann
4	Bootloader	„1“, wenn gerade **kein** Programm ausgeführt wird
5	Programm komplett	„1“, wenn sich im Flash min. 1 Programm befindet
6	Programm korrupt	zur Zeit immer „0“
7	interner Fehler	zur Zeit immer „0“

(04h) „lese Speicheraufbau“
Dieses Kommando erwidert das Telefon mit einer intern vorhandenen Tabelle (z.Z. statisch), die bis zu 32 Einträge enthalten darf. Jeder Eintrag besitzt eine feste Länge von 8 Byte und definiert einen Speicherblock. Alle Einträge werden ohne Trennzeichen direkt hintereinander übertragen. In einem Eintrag sind Typ des Speichers, seine Anfangs- und seine Endadresse enthalten. Da der x186-Controller im Telefon nur einen 20Bit-Adressraum hat, werden die Obersten 4Bit nicht genutzt.

Aufbau des Eintrags:

1. Speichertyp (Aufzählung)*
2. Reserviert für spätere Verwendung
3. High-Byte der 20-Bit Anfangsadresse
4. Mid-Byte der 20-Bit Anfangsadresse
5. Low-Byte der 20-Bit Anfangsadresse
6. High-Byte der 20-Bit Endadresse
7. Mid-Byte der 20-Bit Endadresse
8. Low-Byte der 20-Bit Endadresse

*Der Aufzählungstyp für den Speicher ist wie folgt definiert:

Code	Bezeichnung	Verwendung im Telefon
00h	unbekannt	keine
01h	statisches externes RAM	Ram (der 32Kbyte große SRam-Baustein)
02h	dynamisches externes RAM	keine
03h	µC internes RAM	interner Ram des NEC-Prozessors
08h	(Masken-)ROM	keine
09h	Eprom	z.Z. nicht benutzt
0Ah	E²Prom, r/w	keine
0Bh	Flash-ROM, readonly	z.Z. nicht benutzt
0Ch	Flash-ROM, r/w	der eingesetzte Flashrom, z.Z. fest als 512Kbyte-Block
10h	Memory Mapped Register	interne Funktionsregister des NEC-µC
11h	Memory Mapped I/O	die Register der programmierbaren Logik (EP200)

(10h) „lese Speicherblock aus“
Mit diesem Befehl kann der PC einen Speicherauszug von einem bestimmten Speicherbereich bekommen. Damit lassen sich z.B. RAM-basierte Variablen, dessen Speicherplatz bekannt ist zu einem beliebigen Zeitpunkt kontrollieren. Der Request wird wie folgt aufgebaut (Bytes):

1. Kommandobyte = 10h
2. Speichertyp (Typ des Zielspeichers, entspricht „lese Speicheraufbau“, beim Telefon nur für Test, bei Harvard-Architekturen jedoch sehr wichtig)
3. High-Byte der 20-Bit Anfangsadresse
4. Mid-Byte der 20-Bit Anfangsadresse
5. Low-Byte der 20-Bit Anfangsadresse (entspricht dem x86-Adressraum)
6. Länge des Speicherblockes **minus Eins**

Die Response besteht ausschließlich aus den Binärzeichen des adressierten Speicherblockes. Die Paketlänge ist gleich der Länge der in der Request-Nachricht enthaltenen. Achtung: Ein Auslesen eines einzelnen Zeichens welches zufällig 06h oder 15h (ACK, NAK) bedeuten könnte wird nicht abgefangen – der Master sollte daher möglichst 2 oder mehr Zeichen anfordern, um mißverständliche Nachrichten nicht zu provozieren.

(11h) „gehe in den Programmiermodus“
Durch diesen parameterlosen Befehl wird ein gerade laufendes Programm (z.B. C5-SUP) beendet und SC5BOS in den Ram kopiert. SC5BOS wird dann im Ram gestartet und der Bootloader wird ausgeführt. Dadurch sind bestimmte Kommandos nicht mehr ausführbar und der Ram wird schreibgeschützt. Unter Umständen kann das Telefon einen Wechsel in diesen Modus verwehren (NAK-Antwort). Das passiert immer dann, wenn die Anwendungssoftware sich in einem kritischen Programmteil befindet und per API-Funktion („Unterbrechung durch BL verhindern“) abgesichert wurde. Generell sollte vorher die Anwendungssoftware beendet werden (Kommando F1h).

(12h) „schreibe Block in den Speicher“
Mit diesem Befehl kann der Master einen Block (binaräer Speicherauszug) an eine bestimmt Adresse in Telefon schreiben. Damit lassen sich z.B. Ram-basierende Variablen zu einem beliebigen Zeitpunkt ändern. Befindet sich SC5BOS im Programmiermodus, ist der Ram schreibgeschützt (NAK). Dafür kann jetzt der Flash beschrieben werden wenn der betreffende Bereich leer ist. Ist er nicht leer, so wird mit einem NAK das Schreiben abgebrochen. Der Request ist wie folgt aufgebaut:

7. Kommandobyte = 12h
8. Speichertyp (Aufzählungstyp, entspricht „lese Speicheraufbau“)
9. High-Byte der 20-Bit Anfangsadresse
10. Mid-Byte der 20-Bit Anfangsadresse
11. Low-Byte der 20-Bit Anfangsadresse
12. bis letztes Byte: Nutzdaten

Als Response wird ein ACK oder bei Mißerfolg (Flash nicht leer, Schreibschutz) ein NAK gesendet. Bedingt durch die Dauer einer Flash-Programmierung ist in diesem Fall nicht die übliche TimeOut-Dauer sondern eine berechnete aus normale Timeout-Dauer + Anzahl Bytes * Flash-Programmierzyklusdauer abzuwarten.

(21h) „Updatesequenz einleiten“
Wurde SC5BOS erfolgreich im Ram des Telefons gestartet, kann es sich selbst in den Flash programmieren. Diese Funktion kann mittels Bootloader oder diesem Kommando gestartet werden.
Antwortet das Telefon mit einem ACK, so startet die Lösch- und Programmierroutine des BOS. Nach dem Beenden der Programmierung wird ein weiteres ACK gesendet, um das erfolgreiche Ende des Updates zu signalisieren. Während dieser Zeit dürfen **keine** Kommandos zum Telefon gesendet werden (ca. 3 Sekunden)! Antwortet das Telefon mit einem NAK, so ist der Update-Vorgang fehlgeschlagen. Dies kann durch nicht richtig eingesetzte Bausteine auftreten (Kontaktfehler) und ist leider ein recht fataler Fehler. Man sollte versuchen den Vorgang zu wiederholen (Kommando erneut senden). **Keinesfalls** sollte man das Telefon abschalten, da das ursprüngliche System gelöscht oder beschädigt wurde. Hilft die Wiederholung nicht, so bleibt nur der Ausbau des Flashroms und seine Programmierung im externen Programmiergerät!

(F0h) „Ausführen eines Programmes“
Startet eine Anwendung an (fast) beliebiger Adresse im Adressraum. Man sollte jedoch sicherstellen, das die gestartete Anwendung (oder Subroutine) sich mit einem Far-Return („retf“) beendet.
Als Parameter bekommt das Kommando die Startadresse (20Bit) in High-Mid-Low-Order (wie Cmd 10h) übermittelt. Da der x86 die Adressen segmentiert, wird aus dem oberen 2 Byte (12Bit) das Segment und aus dem letzten Byte der Offset gebildet. Da sich alle NEAR-Sprünge in einer Anwendung auf das Segment beziehen, ist darauf zu achten, das hier das Richtige Segment angegeben wird. Auch bedeutet dies, das der Offset nur in die ersten 256Byte der Anwendung (Subroutine) liegen kann.

(F1h) „Stoppen eines laufenden Programmes“
Dem Anwendungsprogramm wird der Druck auf die ESC-Taste (¤-Taste) vorgegaukelt. Da sich je nach Programmlogik das Programm damit noch nicht beendet (Unterfunktionen z.B.), ist diese gegebenenfalls mehrmals zu wiederholen und mit dem Status-Kommando (03h, Bit4 auf „1“ testen) zu überprüfen, ob der Bootloader aktiviert wurde.

(F2h) „Programmabbruch“
Reagiert ein Programm nicht auf die ¤-Taste bzw. obigen Befehl, so kann das Programm mit F2h „abgeschossen“ werden. Die Einträge der Ereignissroutinen werden sowie der Stack wird zurückgesetzt und er Bootloader wird gestartet.

Port 1 – Debugging

Leider ist der Autor noch nicht dazu gekommen eine Debug-Anwendung zu entwickeln. Auf der Telefonseite ist der Aufwand sehr überschaubar, jedoch ist das PC-Programm sehr umfangreich und erfordert einiges Wissen z.B. über das Objekt-Format.

Port 2 – Text- und Textstatusmeldungen

Das Telefon sendet auf diesem Port Textmeldungen, ohne selbst auf Pakete für diesen Port zu reagieren. Als Textmeldungen sendet der Bootloader z.B. eine Startmeldung. Auch C5-SUP macht davon Gebrauch. Als Entwickler sollte man zu Debugzwecken hiervon Gebrauch machen.

Programmieranleitung

Eine Programmerstellung mit Hilfe der API-Funktionen ist ganz einfach. Zunächst muß man sich drüber im klaren ein, das im Speicher nur 30Kbyte und im Flash (je nach Typ) bis zu 448KByte Platz zur Verfügung stehen. Bei der Programmierung in einer Hochsprache ist dieser Platz sicher recht knapp.

Speichermodell / Programmaufbau

Als Speichermodell sollte Tiny (oder das kleinst mögliche) gewählt. Das Programm benötigt am Beginn einen 16Byte-langen Header, der als erste 6 Zeichen „SC5PRG“ enthält. **Danach folgt ein Word, das die Speichergröße des Datensegmentes aufnimmt.** Dieses Segment befindet sich am Anfang des Programms und wird im vor dem Start in den Speicher kopiert, falls das Programm sich im Flash befindet. Die anderen 8 Zeichen sind beliebig und können für Namen, Versions- oder anderen Text verwendet werden. Ab dem Offset 10h beginnt das SC5PRG. Das Programm wird immer so vom SC5BOS gestartet, das der Offset im Null beginnt (CS:[0] zeigt auf das „S“ von SC5PRG). Die Segmentregister CS und DS zeigen beim Start auf das (gleiche) Programmsegment wenn das Programm sich im Ram befindet, ansonsten Zeigt DS auf das Ram (typ. 80h) und CS auf das Flash (z.B. 8000h). Beispiel:

6 Byte	SC5PRG	Programmkennung
2 Byte	42h 03h	Länge der initialisierten Daten = 0342h Bytes (ab 0)
8 Byte	SUP v2.2g	Programmname (ASCII-Zeichen)
2 Byte	E9h 85h	Sprungbefehl „jmp main“

Durch die Kopierfunktion sind alle Variablen, die sich am Anfang innerhalb der initialisierten Daten befinden gleich verfügbar. Das Datensegment ist entsprechend gesetzt. Der Speicherplatz für nicht initialisierte Daten muß das Programm selbst bestimmen. Befindet sich das Programm im Ram, so können die Daten im nächsten freien Segment nach dem Code untergebracht werden. Beim Start aus dem Flash ist unmittelbar nach dem initialisierten Bereich Platz. In jeden Fall ist ein extra Segmentregister (ES) nötig.
Da zur Assemblierungszeit nicht feststeht, in welchem Segment das Programm geladen wird (kann ja auch in den Flash geschrieben werden), sind **alle Aufrufe** wie „mov ax, seg Var“ **ungültig**, da die Segmentkostanten zur Assemblierungszeit gesetzt werden. Beachtet man dies nicht, kann das Programm nur noch an eine Adresse in das Flash geschrieben werden. Benutzt man einen Hochsprachen-Linker und erstellt eine COM- oder EXE-Datei, so kann diese in ein SC5PRG umgewandelt werden. Bei EXE-Dateien muß man jedoch bei der Konvertierung ebenfalls schon die richtige Adresse im Flash angeben, jedoch hat man dann den Vorteil, sehr große Programme schreiben zu können (>30Kbyte, >64Kbyte). Ein Programm „SC5-EXE-Loader“* der diese Konvertierung vornimmt ist geplant.

* Der SC5-EXE-Loader existiert noch nicht, da bisher kein Bedarf an so großen Programmen existiert.

Aufruf von API-Funktionen

Das Application-Interface wird über den Interruptvektor 22h angesprochen. Die Funktionsnummer wird im AH-Register übergeben. Parameter sind funktionsspezifisch in anderen Registern zu übergeben. Durch den Aufruf des Int 22h wird das BX-Register **in jedem Fall zerstört** und je nach Funktion werden auch andere Register verändert. Am häufigsten wird das ES-Register von der API benutzt. Das DS-Register bleibt jedoch immer unangetastet. Eine Übersicht über das API folgt im nächsten Kapitel.

Programmieren von Ereignisbehandlungsroutinen

Eine Ereignisbehandlungsroutine ist eine einfache Funktion, die den Programmablauf an beliebiger Stelle unterbrechen kann und mit den evtl. übergebenen Daten eine kurze „Behandlung“ durchführt. Daher werde dies auch kurz „**Handler**“ genannt.
Im Telefon stehen 16 verschiedene Quellen für Handler bereit, 10 davon sind für den Datenempfang über die serielle Schnittstelle reserviert (die „Ports“ – siehe PCP-Kapitel).

Ein Handler ist für den Programmierer erst einmal eine normale Funktion, die mit einem Far-Return („retf“) endet. Bei der Hochsprachenprogrammierung ist darauf zu achten, das der Compiler die Funktion nicht weg optimiert (z.B. „static“ verwenden). Da das Datensegment initialisiert wurde, gibt es im Grund nichts weiter zu beachten. **Ausnahme:** Da bei allen PCP-Empfangs-Handler jedoch DS:[SI] auf den empfangenen Block zeigt und ES auf das eigene Datensegment, muß hier der Programmierer aufpassen, nicht ins falsche Segment zu schreiben.
Um einen Handler beim SC5BOS zu registrieren, wird einfach die API Funktion „setintvec“ mit der passenden Funktionsnummer (Interruptnummer) verwendet . SC5BOS erkennt, das es sich um einen Handler handelt und speichert das ES-Register zusätzlich in dem Reservierten Bereich ab, der auf die Handlervektoren folgt. Bei jedem Handler-Aufruf stellt SC5BOS DS anhand dieses Bereiches wieder her (Ausnahme bei PCP beachten). Diese Funktionalität hat einen Nachteil: Ein von einem anderen Programm belegter Handlervektor **kann nicht** einfach **temporär überschrieben** und wiederhergestellt werden. Diese Funktionalität ist jedoch hoffentlich nicht nötig – eine Anwendung im C5 sollte z.B. immer einen eigenen PCP-Port benutzen. Auch mehrere um Tastatureingaben konkurrierende Programme werden hoffentlich nicht die Regel im C5!
Ein Handler wird mit der Funktion 0Bh „entferne Handler“ wieder sauber entfernt (incl. Datensegment-Wert).

Eine Übersicht über die verfügbaren Ereignisbehandlungsroutinen folgt im übernächsten Kapitel „Ereignisse“

Application Interface

Das Basis-Funktionen werden über mehrere Software-Interrupts aufgerufen. Die Parameterübergabe erfolgt über Register (analog DOS). Die Nummer der aufzurufenden Funktion wird im AH-Register übergeben, ein Byte-Parameter immer im AL-Register. Die Rückgabe erfolgt im AX-Register Funktions-spezifisch. Allgemein gilt AX=0 als fehlerfreie Ausführung. Zeiger auf Daten werden immer als DS:[SI] übergeben, wobei DS das Segment und SI den Offset enthält. Längen sind meistens im CL oder CX untergebracht. Zeiger auf zu füllende Datenblöcke werden als ES:[DI] übergeben. Die Funktionen des SC5BOS sind in 5 Funktionsgruppen unterteilt:

1. Steuerungsfunkionen SC5BOS
2. BF-Bus – Zugriff und Abfrage
3. Serielle I/O-Funktioenen (Ser0 / I²C-Bus)
4. Informations- und Konvertierungsfunktionen
5. DSP-Funktionen

Nachfolgend sind die implementierten und zukünftigen Funktionen tabellarisch aufgeführt. Der I-Status (Implementierungsstatus) gibt an, ab welcher SC5BOS-Version die Funktionen enthalten sind. Ist das Feld leer, so ist die Funktion nicht implementiert.

0xh – Steuerungsfunktionen

Funktion	AH	AL	Beschreibung	I-Status
Programm beenden oder Gerät abschalten	00h	-	Schaltet das C5 aus (INT3Bh-Aufruf)	v0.14
	01h	-	Hardware-Reset des C5 (INT3Bh-Aufruf)	v0.14
	02h	-	Softwarereset normal	v0.14
	03h	-	Softwarereset, Start des Bootloaders	v0.14
	04h	-	Startet Bootloader ohne Reset	v0.14
Interruptvector / Behandlungs-routine setzen	05h	Vec	*IN:* AL=Interruptvektor, DS:[SI]=INT-Routine, bei Behandlungsroutine ES = benutztes Datensegment	v0.15 Ä0.17b
Stack in den SRAM verlagern	06h	-	Die Funktion kopiert einmalig den Stack an das obere SRAM-Ende (für Programme, die mehr als 64Words Stack benötigen).	v0.14
Stackwarnung, Fehler	07h	00h	Stackwarnung ausschalten	v0.14
		01h	Stackwarnung einschalten (**default**)	
Gerät abschalten steuern	08h	00h	Ausschalten erlaubt (**default**)	v0.14
		01h	Ausschalten verboten (Taste keine Wirkung)	
Unterbrechung durch BL	09h	00h	Unterbrechungen zulassen (**default**)	v0.14
		01h	Unterbrechungen verbieten (für kritischen Code)	
Ducklänge der ¤-Taste bis zum Abschalten.	0Ah	Val	*IN:* AL = Länge (x10ms), die der Benutzer die Aus-Taste gedrückt halten muss.	v0.14
Behandlungsroutine entfernen	0Bh	Vec	*IN:* AL = Vektor der Behandlungsroutine	v0.17b
Zündungs-Timer einstellen	0Ch	Tmr	*IN:* AL = Anz. Minuten, die das C5 noch an bleibt	v0.30
Warten (passives Warten)	0Eh	-	*IN:* CX: Wartezeit in (CX*10ms)	-
Warten (aktive Warteschleife)	0Fh	-	*IN:* CX: Wartezeit (CX=256 **à** 30ms Warten)	v0.14

Die Beenden- und Reset-Funktionen
Das BOS verfügt über eine Reihe von verschiedenen Reset-Funktionen. Neben der Beendenfunktion die das C5 abschalten läßt, gibt es nur noch 2 weitere verschiedenartige Funktionen, da 01h, 02h und 03h alle einen Software-Reset auslösen (Hardware-Reset ist beim C5 im Gegensatz zu anderen Telefonen nicht möglich).
Bis auf Funktion 04h lösen die Funktionen die Aufruf der Abschalt-Behandlungsoutine (Int 3Bh) aus. Hier sollte der Programmierer „aufräumen“ und sämtliche eigenen Handler entfernen, die Display-Symbole löschen und die verwendete Hardware zurücksetzen: z.B. **SENDER AUS**!

Interrupts und Handler setzen und wieder entfernen
Prinzipbedingt kann jedes Programm selbst in die Interrupt-Vektortabelle des x86-Prozessors schreiben. Da hier jedoch neben „normalen“ Vektoren die Vektoren und Datensegmente der Behandlungsroutinen sowie die vom System benutzten Vektoren geschpeichert sind, sollte man die Funktion 05h benutzen. Diese Funktion „erkennt“ anhand der in AL übergebenen Vektornummer auch, ob es sich um einen Handler handelt und speichert dann das Datensegment des Programmes (ES-Register) in einem reservierten Bereich der Verktortabelle. Die Adresse der Routine wird „normal“ über das DS und SI Register übermittelt (DS = Code-Segment, SI = Offset der Routine).

Mit der Funktion 0Bh wird ein Handler (Nummer im AL-Register) wieder entfernt, d.h. die Adresse wird auf eine leere Dummy-Routine innerhalb SC5BOS gesetzt. Das Datensegment wird auch auf das von SC5BOS geändert.

Der Stack
Der NEC-Prozessor besitzt einen kleinen internen RAM, der für verschiedene Zwecke benutzt werden kann. Neben weiteren Registerbänken und einem Bereich für MacroServices bleibt ein „Rest" von 64 Words, die im Telefon für den Stack verwendet werden. Der Vorteil: Der Ram kann von Adresse 2048 (800h) an komplett benutzt werden. Werden jedoch umfangreiche Programme in einer Hochsprache entwickelt, so reicht dieser Platz nicht. Da besonders lokale Variablen auf dem Stack landen wäre dieser schon bei einer Verschachtelungstiefe 2-3 voll. Daher bietet das SC5BOS die Möglichkeit, den Stack in den Ram zu verlagern (Funktion 06h). Er wird an das obere Ende kopiert und wächst nach unten. Eine Umkehrung dieser Funktion ist nicht möglich.
Entwickelt man ein kleines Programm oder in Assembler, so kann der Stack im internen Ram verbleiben. Um Problemen mit dem Stack zu vermeiden überprüft SC5BOS ständig, ob der Stack zu knapp wird und schaltet das Telefon gegebenenfalls aus, um Fehlfunktionen zu vermeiden. Dieser Test kann nur für den internen Stack durchgeführt werden. Es wird mit der Funktion 07h ab- oder eingeschaltet (minimal mehr Performance, wenn deaktiviert).

1xh – BF-Bus - Zugriff und Abfrage

Funktion	AH	AL	Beschreibung	I-Status
BF-Bussteuerung	10h	00h	BF-Bus-Behandlung abschalten (**à** INT 3Ah wird nicht ausgeführt)	v0.14
		01h	BF-Bus-Behandlung einschalten (**à** INT 3Ah ist aktiviert, **default**)	
Sende Paket zum Hörer	11h	-	*IN:* Paket in DS:[SI] mit der Länge CL,	v0.15
QuittungsTon	12h	-	Gibt einen leisen einzelnen Ton aus	v0.30
FehlerTon	13h	-	Gibt einen leisen Doppelton aus	v0.30
Warte auf Hörer-Taste	14h	-	*OUT:* AL = Tastencode	v0.15
Letzte Hörertaste abfragen	15h	-	*OUT:* AL = Tastencode, AH = 0h, wenn Taste neu	v0.15
Setze Wiederholrate	16h	Val	*IN:* AL = Tastenwiederholrate in 10ms	v0.15
Setze Startverzögerung	17h	Val	*IN:* AL = Verzögerungszeit erste Wiederh. (10ms)	v0.15
Sende Paket an Gerät	18h	Adr	*IN:* AL = Geräteadresse, DS:[SI] = Paket mit Länge CL	v0.32
Acknowlegde senden	19h	Adr	*IN:* AL = Geräteadresse	v0.32

Die BF-Bus-Funktionen ersetzten die bei PCs typischen Keyboard und Bildschirmfunktionen. Standardmäßig behandelt SC5BOS den BF-Bus und verarbeitet die Datenpakete vom Hörer. In Einzelfällen kann es vorteilhaft sein, die komplette Steuerung selbst zu programmieren, um etwa statt der langsamen 19k2-Verbindung eine eigene schnellere Betriebsart auf der BF-Busleitung zu implementieren (z.B. für grafikfähige Bedienteile). Mit der Funktion 10h kann die Behandlung deshalb ab- und wieder eingeschaltet werden.
Mit der „Hauptfunktion" 11h sendet man ein Datenpaket zum Hörer. Das Format und der Aufbau der Pakete ist im Kapitel „BF-Bus" beschrieben. Neben dieser gibt es noch 2 Funktionen um öfter benötigte akustische Signale auszugeben.

Mit den Funktionen 18h und 19h kann ein beliebiges „Gerät" z.B. die Optionsmodule angesprochen werden. Die Funktion 19h ist dabei ein Spezialfall der 18h, da ACK öfters benötigt wird.

2xh – Serielle I/O-Funktioenen (Ser0 und I²C-Bus)

Funktion	AH	AL	Beschreibung	I-Status
Serial0: Betriebsmodus	20h	?	? Auswahl, ob Chipkarte oder externe Schnittstelle	-
Serial0: Baudraten Ser0	21h	Val	*IN:* AL=Aufzählung Baudrate	-
Serial0: Sende Zeichen	22h	Val	*IN:* Zeichen, das direkt auf TxD0 geschickt wird	-
Serial0: Sende Paket	23h	-	*IN:* DS:[SI], CX = Paket, analog zu 22h	-
Serial0: Send-PCP_Short	24h	Port	*IN:* AL=Port, DL=Zeichen	v0.14
Serial0: Send-PCP-Long	25h	Port	*IN:* AL=Port, DS:[SI], CX = Paket	v0.15
Serial0: Sende zur Chipkarte	26h	-	*IN:* DS:[SI], CX = Paket	-
Serial0: Lese von Chipkarte	27h	-	*IN:* ES:[DI] = Buffer, CX = ReadLength (max)	-
I²C: Abbruch ausgeben	28h	Char	*IN:* AL = Zeichen, das gesendet wird	v0.15c
I²C: Daten ausgeben	29h	Adr	*IN:* AL = I²C-Geräteadresse, DS:[SI], CX = Paket	v0.15c
I²C: Daten auslesen	2Ah	Adr	*IN:* AL = I²C-Geräteadresse, ES:[DI] = RXBuffer, CX = Anzahl auszulesender Bytes	v0.15c
I²C: Kombi: Ausgabe, Einlesen	2Bh	Adr	*IN:* AL = I²C – Geräteadresse (Schreibadr.) DS:[SI], CL = Paket für Aussendung, ES:[DI] = RXBuffer, CH = Anzahl auszulesender Bytes	v0.15c
EEProm: Daten schreiben	2Ch	Adr	*IN:* DS:[SI] = Zeiger auf die Daten, CL = Anzahl der Daten, AL = Word-Adresse der ersten Speicherzelle.	v0.30
EEProm: Daten lesen	2Dh	Adr	*IN:* ES:[DI] = Buffer für die Daten, CL = Anzahl, AL = Word-Adresse der ersten Speicherzelle	v0.30

Hinweis: Die API-Funktion I²C Acknowledge Polling wurde ersatzlos entfernt, da sie nicht benötigt wird!

Die PCP-Funktionen

Über die beiden Funktionen 24h und 25h kann das Anwendungsprogramm Daten an einen angeschlossenen PC senden. Die Daten werden vorher mit einer Prüfsumme versehen **à** siehe Kapitel PCP am Ende der Doku.
Über den Port, der in AL übergeben wird steuert man die Anwendung auf dem PC an. dadurch ist ein simultaner Betrieb von z.B. Telefonprogrammierer, MessageViewer und SUP-CP möglich. Ein-Zeichen-Nachrichten können einfach mit der Funktion 24h versendet werden. Das Zeichen wird in DL übergeben, der Zielport in AL. Alle anderen Nachrichten werden per Funktion 25h übermittelt.
Da SC5BOS das Paket von DS:[SI] vor dem Versenden kopiert, kann der Inhalt unmittelbar nach Funktionsaufruf verändert werden. Die Übertragung läuft asynchron im Hintergrund. Ist der Sendebuffer voll, wird auf das Freiwerden gewartet. Aufgrund des knappen Ram des SC5BOS (1Kbyte) kann der Sendebuffer nur ein Paket vorhalten. Spätere Versionen von SC5BOS werden jedoch den Platz flexibler nutzen, so das mehrere kleine Pakete vorgehalten werden können, was z.Z. noch nicht geschieht.
Das Empfangen von Paketen kann generell nur durch eine Ereignisroutine (30h-39h) erfolgen. Ein Pollen auf Pakete, wie es oft noch üblich ist, hat der Autor bewußt nicht vorgesehen. Die Rechenleistung des NEC verbietet einen großzügigen Einsatz von Polling.

I²C-Funktionen

Die I²C-Anbindung wurde in den Siemens-Telefonen über einen ASIC-IC realisiert. Im C5 ist dies der EP200. Neben dem I²C hat dieser jedoch noch viele weitere Aufgaben.
Da die Kommunikation per I²C nicht so einfach ist, wie mit einem „echten“ I²C-µC. Aus diesem Grunde ist dieser Teil im SC5BOS realisiert. Neben Ausgabe und Einlesen von Daten (der NEC-µC ist der einzige Master) von PLL und AD-Wandler wurden beide Funktionen in der Funktion 2Bh kombiniert. Dies ist für den AD-Wandler sehr sinnvoll, um gleich den zu Kanal weiter schalten zu können.

Spezialfall I²C-EEProm

Das serielle I²C-Eprom ist auch nicht leicht anzusprechen – der Autor verwendet eigene I²C-Routinen, die nur für den EEProm Zugriff gedacht sind. Das Auslesen eines Bytes stellt noch keine großen Anforderungen und könnte auch durch die I²C-Kombi-Funktion erfolgen (jedoch ist das nicht Datenblatt-konform).
Beim Schreiben muß darauf geachtet werden, das immer nur 2Byte am Stück geschrieben werden. Solange der EEProm schreibt antwortet er nicht auf I²C-Kommandos (ACK-Polling). Das komplette Schreibverhalten ist daher in der Funktion 2Ch implementiert. Der Programmierer braucht sich darum keine Gedanken mehr zu machen.

3xh – Informations- Konvertierungsfunktionen

Funktion	AH	AL	Beschreibung	I-Status
Informationen über SC5BOS	30h	-	*OUT:* Version in AX (Hi.LOW)	v0.14
	31h	-	*OUT:* Einschaltdauer in AX, BH, BL (H, M, S)	v0.14
	32h	-	Fehlerzähler des BF-Busses (Frame, Overrun **à** AX)	v0.14
	33h	-	Fehlerzähler der ext. Schnittstelle (analog 02h)	v0.14
Flashgröße bestimmen	34h	-	*OUT:* AX = Flashgröße in Kbyte	v0.15
Flashhersteller und Device-ID	35h	-	*OUT:* AH = Hersteller ID, AL = Device ID	v0.15
Interruptvector lesen (zum Sichern z.B.)	36h	Nr	*IN:* AL = Interruptvektor *OUT:* DX:AX = Adresse der Interruptbehandlungsfunktion (Seg:Ofs)	v0.15e
Zahl **à** ASCII (ES:[DI])	39h	Val	*IN:* AL = Tahl, Umwandlung in 3stellige Dezimal-darstellung. *OUT:* Ascii **à** ES:[DI], DI unverändert	v0.9a
	3Ah	Val	*IN:* AL = Tahl, Umwandlung in 4stellige Dezimal-zehnteldarstellung: "12.3" *OUT:* Ascii **à** ES:[DI], DI unverändert	
	3Bh	-	*IN:* DX = Tahl, Umwandlung in 5stellige Dezimal-darstellung. *OUT:* Ascii **à** ES:[DI], DI unverändert	
	3Ch	-	*IN:* DX = Tahl, Umwandlung in 6stellige Dezimal-hundersteldarstellung: "123.45" *OUT:* Ascii **à** ES:[DI], DI unverändert	
	3Dh	Val	*IN:* AL = Zahl, Umwandlung in 2.-Hexdarstellung *OUT:* Ascii **à** ES:[DI], DI = DI + 0	v0.14
	3Eh	-	*IN:* DX = Zahl, Umwandlung in 4.-Hexdarstellung *OUT:* Ascii **à** ES:[DI], DI = DI + 4	
	3Fh	Val	*IN:* AL = Zahl, Umwandlung in 8.-Binärdarstellung *OUT:* Ascii **à** ES:[DI], DI = DI + 8	

Verwendung der Informationen über SC5BOS
Die Anwendung sollte in der Startroutine immer die Version von SC5BOS (Funktion 30h) ermitteln, um sicherzugehen, das alle API-Merkmale vorhanden sind. Ist die Version zu niedrig sollte eine Meldung auf den Hörer erscheinen und das Programm nach Tastendruck beendet werden. Einschaltdauer und Fehlerzähler können für ein Diagnoseprogramm benutzt werden. Z.Z. werden diese Funktionen in C5-SUP nicht benutzt.
Der Flashtyp und die DeviceID liefern leider keine gültigen Werte, da die Ausleseroutine leider nicht mehr ins Ram paßt. SC5BOS hat für seine Funktion nur 1Kbyte nach der Interrupttabelle zur Verfügung.

Konvertierungsfunktionen
Die Funktionen 39h bis 3Fh dienen vor allem dem Assembler-Programmierer um sich die in Assembler umständliche Implementierung zu ersparen. Die Funktionen 39h bis 3Ch sind dabei so neu, das sie von C5-SUP noch nicht benutzt werden (hier ist der gleiche Quelltext verwendet worden).

4xh – DSP-Funktionen

Funktion	AH	AL	Beschreibung	I-Status
Reset DSP	40h	-	Führt einen Rest per Reset-Signal am DSP durch	0.20
Übertrage DSP-Programm	41h	-	*IN:* DS:[SI], CX = Datenblock, der das DSP-Programm enthält.	0.20
Sende zum DSP	42h	-	*IN:* DX:CH = 24bit-Wort das der DSP erhalten soll.	0.20
DSP-Kommando	43h	Cmd	*IN:* AL = Kommando (0 bis 11), welches das DSP-Programm ausführen soll. DX:CH = 24bit-Wort das der DSP vorher erhält.	0.20

Damit im Siemens-Telefon überhaupt ein Ton hörbar wird, muß das Audiosignal in beiden Richtungen durch den DSP. Nach dem Starten des C5 ist sein Programmspeicher jedoch leer. Ein Bootrom im DSP sorgt dafür, das man das Programm über das Host-Interface heineinladen kann. Dies passiert mittels Funktion 41h.
Im Betrieb ist es nötig dem DSP-Programm Anweisungen und Daten zu übermitteln. SC5BOS stellt dafür die Funktionen 42h und 43h bereit. Für weitere Information zu diesem Thema siehe Kapitel Host-Interface, DSP-UsersManual und das DSP-Program C5Voice (auf der Downloadseite zu finden).

Ereignisse

In dem Telefon gibt es eine Reihe von asynchron zum Programmablauf auftretenden Ereignissen. Diese können vom Programm auch asynchron behandelt werden. Im SC5BOS ist eine umfassende Unterstützung dieses Konzeptes realisiert.
Die Ereignisbehandlungsroutinenen (kurz Handler) sind normale Prozeduren (als FAR compiliert), die durch SC5BOS aufgerufen werden. (vor Version 0.18 waren Handler-Routinen als Soft-Interrupts implementiert.) Die Routinen werden analog „normalen" Interrupt-Behandlungsroutinen mit dem „set Interruptvector"-Befehl gesetzt. Jedoch muss SC5BOS vor jedem Handler-Aufruf wissen, welches Datensegment das passende ist. Daher wird SC5BOS im ES das Datensegment zusätzlich übergeben. Die Datensegmente werden im Anschluß an die Handler-Vektoren in die Interruptvektortabelle geschrieben, daher können keine Soft-Ints 40h-4Fh benutzt werden!
Diese Routine muß mit einem „retf"-Befehl abgeschlossen sein (unter Hochsprachen erreicht man das mit einer FAR-Direktive. Es sind folgende Register gesichert.:

- AX, BX, CX, DX, SI, DI, DS, ES
- **Nicht** gesichert dagegen: SS, SP, BP

Übersicht

INT	Funktion		Beschreibung	I-Status
30h . . . 39h	**well-known-Ports des PCP-Protokolles**	**00h**	KommunikationsINT (Rx) des SC5BOS	v0.1
		01h	KommunikationsINT (Rx) des DebuggingModuls	-
		02h	Reserviert (Textmeldungen) Empfangshandler bei Nicht-PCP	v0.11 v0.9a
	Ports geräteabhängiger Kanäle (SC5COM)	03h	C5-Gerätesteuerung, Fernbedienung	-
		04h	KommunikationsINT (Rx) für 1k2 PR	-
		05h	KommunikationsINT (Rx) für 9k6 PR	-
3Ah	INT-Handler des BF-Busses*		Ein Paket ist auf dem BF-Bus empfangen worden (gilt nicht für Tastenbetätigungen am Hörer)	
3Bh	Abschalt-Handler		Das C5 wird abgeschaltet (kein Watchdog mehr)	v0.12b
3Ch	Tastenbetätigung am Hörer*		*PARAM:* AL=Tastencode, AH= Betätigungszähler	v0.12b
3Dh	periodischer Handler (Timer)		keine Parameter, Aufruf alle 50ms (10Hz)	v0.17
3Eh	DSP-Handler (Receive)		*Param:* DX:CH 24bit Word vom DSP	v0.30
3Fh	Alarm-Handler		Fehler in AL: Bits: (1) Temp_PA, (0) Akku_Low	v0.30
40h bis 4Fh	Reserviert, siehe Text			

* **à** Wenn die Anwendersoftware den INT 3Ah benutzt, so ist zu bedenken, das Tastenbetätigungen nicht im INT 3Ah abgefangen werden können. Dafür muss die Software zusätzlich den INT 3Ch auf eine eigene Behandlungsroutine setzen.

Bis auf die PCP-Handler und dem BF-Bus-Handler werden die Daten in den Registern direkt übergeben.

PCP-Empfang

Über die Ereignisse mit den Nummer 30h bis 39h wird das Programm über eintreffende Datenpakete vom PC informiert. Dabei wird jedoch Port 0 (Int 30h) bereits vom SC5BOS verwendet und sollte nicht verwendet werden. Da die Pakete über das Registerpaar **DS:[SI]** übergeben werden, muß in der Routine vor dem Zugriff auf Variablen das Datensegment wieder zurückgesetzt werden. Beispiel:

```
mov   BX, DS
mov   DS, ES
mov   ES, bx
mov   Variable, ES:[SI] ; Zugriff
```

Verwendet man jedoch nur Assembler, so genügt es im Handler die globalen Variablen über das ES-Register anzusprechen:

```
mov   ES:Variable, DS:[SI]
```

Die Datenlänge des Paketes wird in CX übergeben. Die Prüfsumme, sowie die zusätzlichen Bytes des PCP-Protokolls werden nicht übergeben.

Für die Daten gilt: Das übernächste eintreffende PCP-Paket **überschreibt die Daten**. Die Behandlungsroutine sollte sich mit der Auswertung nicht zuviel Zeit lassen. Nach minimal 6 Bytezeiten = 60Bitzeiten (bei 57k6 ca. 1ms) können die Daten bereits ungültig werden. Möchte man länger auf die Daten zugreifen können, so sollte man sie zuerst kopieren. Zur Steuerung von C5-SUP ist dies noch nicht nötig, die Parameter gleich in Register geladen werden.

BF-Bus-Handler

Im Gegensatz zum PCP-Handler ist beim BF-Bus-Handler das Datensegment DS bereits richtig gesetzt. Die eingetroffenen Daten befinden sich hier in **ES:[BX]**. Eine Länge wird nicht übergeben, da sie im Paket enthalten ist (die Daten enthalten das komplette Paket, wie es auf dem Bus gesendet wurde). Der Aufbau des Paketes ist im Kapitel „BF-Bus“ erläutert.

Eine Sonderstellung nehmen die Hörertasten ein:
SC5BOS filtert diese Pakete heraus und sendet auch das ACK zum Hörer. Die Tastenbetätigungen können über den Handler 3Ch behandelt werden oder über die API-Funktionen 15h und 16h abgefragt werden. Diese entsprechen dem „getkey“ und „readkey“ Funktionen eines PC-Programmes.

Auch für den BF-Bus-Handler gilt: Die Daten werden nach 3 weiteren empfangenen Paketen überschrieben. Es ist jedoch lange nicht so kritisch, wie beim PCP-Empfang.

Abschalt-Handler

Aus Sicherheitsgründen überwacht das Betriebssystem den Abschaltvorgang. Ein Unterdrücken dieser Funktion sollte auch nur mit großer Vorsicht verwendet werden (d.h. nicht während der Entwicklung). Damit die Anwendung eine Chance bekommt wichtige Daten in das EEProm zu sichern und „aufzuräumen“, wird sie über diesen Weg über das „nahende Ende“ informiert. Ab diesem Zeitpunkt sind noch ca. 0,8s Zeit für wichtige Funktionen. Die Abschaltmeldung wird vom SC5BOS nach Beenden dieser Funktion dargestellt.

Tatenbetätigungen

Für jeden Tastendruck generiert SC5BOS einen Tastencode (im AL-Register). Wird die Taste niedergehalten, so generiert SC5BOS nach einer Verzögerung weitere Betätigungen automatisch (Intervall einstellbar). Die Tastencodes sind im Kapitel „Tastaturcode des Standard-Hörers“ aufgelistet. Der Tastenzähler (AH-Register) zählt einfach die Betätigungen mit.

Timer

Für alle periodisch auftretende Funktionen kann der Programmierer auf diesen einfache Timer zurückgreifen, der alle 50ms (10Hz) aufgerufen wird. Hier können z.B. die AD-Werte ausgelesen werden oder mit einem Vorteiler (z.B. 4) der Displayinhalt erneuert werden.

DSP-Daten entgegenehmen

Nicht immer reicht es den DSP Kommandos zu schicken. Im Fallbeispiel VOX-Steuerung muß auch das DSP-Programm fähig sein, der Anwendungssoftware Daten zu schicken. Da der DSP intern mit 24bit-Wortbreite arbeitet wird das empfangene Datum als DX:CH (High = DH, Mid = DL, Low = CH) der Routine zur Verfügung gestellt. Was wie in diesem Wort codiert ist, bestimmt das DSP-Programm.

Alarmbehandlung

Mit dem Alarm-Handler kann das Anwendungsprogramm auf kritische Betriebszustände reagieren, ohne die Eingänge zu pollen. Im einfachsten Fall genügt dem Benutzer eine Warnmeldung zu präsentieren. Solange der Alarmzustand anhält wird die Routine sekündlich wiederholt aufgerufen.
Eine guter Ansatz ist z.B. bei Akkualarm ein Abschalt-Zähler herunter zu zählen, der beim Ausbleiben des Alarms (Timer) wieder zurückgesetzt wird.
Der Fehlergrund ist als Bitmuster in AL codiert: Bit0 = Akku hat Unterspannung, Bit1 = PA-Temperatur zu hoch (Fehlanpassung etc.). Beide Fehler können gleichzeitig auftreten.
Hinweis:
Der Akku-Alarm erlischt nicht unbedingt gleich nach Anschluß eines Netzteiles (wenn der Akku sehr schwach ist), der Alarm sollte beim Erkennen eines Netzteiles trotzdem unterdrückt werden.

Informationen zur C5-Hardware

In diesem Abschnitt ist in Stichpunkten zusammengetragen, was es so übers C5 als Entwickler wissen sollte. Leider lassen sich nicht besonders viele Informationen zur Technik des C-Netzes auffinden. Hier ist eine kleine aber gute Zusammenfassung:

http://www.e-online.de/sites/kom/0408091.htm

Aufgrund der kombinierten Daten- und Sprachübertragung gibt es auch in der C5-Hardware ein paar Besonderheiten (näheres siehe Codec-Signalweg).

Speicheraufteilung

Der Adressraum des NEC-Microcontrollers beträgt 1024kByte. Davon wird im C5 nur die unteren 64Kbyte und die obersten 256Kbyte (Originalsoftware, nach dem Umbau 512Kbyte) benutzt. Das C5 besitzt 32Kbyte statischen RAM an der Adresse 0000h bis 3FFFh. Hier befinden sich die Interruptvektortabelle des x86-kompatiblen NEC's, (1Kbyte) sowie beim Betrieb von SC5BOS ein weiteres KByte Daten. Die weiteren 30Kbyte sind vollständig durch Programme nutzbar, wenn der Stack im Internen RAM des NEC belassen wird. Ansonsten sollte an der oberen RAM-Grenze der Stack-Bereich nicht beschrieben werden. In dem Bereich von 4000h bis 7FFFh befinden sich verschiedene Speichereinblendungen:

- **C000h - CFFFh**: Host-Interface des DSP (Register an C101h, C202h usw.)
- **D000h - D001h**: Paralleler DAC für AFC und TXLEV_REF (Sendeleistung)
- **E000h - E01Fh**: Speichereinblendung auf Register des EP200 – welche genau was bewirken ist noch Forschungsarbeit. Der Autor bittet Jeden, der neue Infos zum EP200 hat, ihm Diese mitzuteilen.
- **FE00h - FEFFh**: Interner RAM des NEC-µC (der Bereich wird hier sowohl in der Originalsoftware, als auch bei SC5BOS eingeblendet, ist jedoch in 4K-Schritten wählbar)
- **FF00h - FFFFh**: Special-Function-Register des NEC-µC (Einblendung mit int. RAM gekoppelt)

Danach folgt erst mal eine Weile nix. Ab der Adresse 80000h ist dann der Flashrom eingeblendet. Glücklicherweise generiert der EP2000 sowohl /CS als auch /WE Signale für den Bereich von 80000h bis BFFFFh. Im C5 wäre eigentlich keine Veranlassung für ein solches Verhalten, da der EPROM erst ab Adresse C0000h benutzt wird (von 80000h bin BFFFFh spiegelt er sich).

Der EP200

Leider ist der GAL mit der Bezeichnung EP200 immer noch ein großes Rätsel für sich. Er ist gehört von seiner Funktion weder zur reinen „Glue-Logic" noch zur reinen Signalverarbeitung. Dieser Baustein vereinigt verschiedenste Funktionen in sich. Hier eine kurze Aufzählung:

- aus Quarzoszillator werden verschiedene Takt- und Clocksignale erzeugt: µC, Combo-Clock-Signale etc. (bis auf µC sind alle Signale ein-/abschaltbar)
- I²C realisieren.
- Bereitstellen von zusätzlichen I/O-Ports.
- „multiplexen" der zweiten seriellen Schnittstelle.
- C-Netz-Datenpakete (zwischen den komprimierten Audiodaten) herausfiltern und verarbeiten (hier wurde der DMA des NEC benutzt).
- ... (bestimmt noch mehr Features)

Laut Aussage eines nicht näher bekannten Insiders:
„...habe mit einem Entwickler vom FP100 ASIC (EP200 , Entwicklungsprojekt 2000) gesprochen. In diesem Baustein wird nur die Signalverarbeitung vom C-Netz gemacht. Als Komprimierung, Dekomprimierung, ausfiltern der Daten usw. aber keine NF Bearbeitung. Da von den IC nur 2 Schüsse gemacht wurden ist er Fehlerhaft und stürzt öfters ab. Diese Probleme wurden Softwaremäßig kompensiert." (Zitat Ende)

Clockgenerierung

folgt noch...

EP200-Ports

Mapped Adresse	Funktion	Beschreibung
E000h	EP200-Port	Bit6 = BF-Bus, Bit4 = NF-Enable (Lautsprecher)
E001h	Combo-Steuerung	
E002h	I²C Data	s.u.
E003h	I²C Control	s.u.
E004h	Steuerung / ser. Schnittstelle	evtl. sind hier Steuerbits für Kartenzugriffe
E010h	Abtastintervall Combo1	174 (0Aeh) ergeben die (gewünschten) 104µs für 9600Sps
E011h	Steuerung FSR/FSK … (?)	085h schaltet den FSx-Takt vom Combo1 ein, welche Bits was machen -> ? Bit5 schalten C5 jedenfalls ab – Fehler?
E01Ah	Umschaltung extern/Karte	mit 0Ch kann ser. Schnittstelle auf Extern geschaltet werden.

folgt noch...

Übersicht des Speichermapping

folgt noch...

Portfunktionen des NEC und des EP200

folgt noch...

Serielle Schnittstellen

Der NEC-Microcontroller verfügt über 2 unabhängige fullduplex UARTs die synchron oder asynchron betrieben werden können. An dem ersten Port (Ser0) ist im C5 der BF-Bus angeschlossen. Der zweite Port (Ser1) ist dagegen nur mit dem EP200 verbunden. Außerdem ist die CCData-Leitung des Kartenlesers mit der RXD-Leitung verbunden. Über Controlregister im EP200 kann die Benutzung von Ser1 gesteuert werden. Dabei kann bestimmt werden, ob die Schnittstellen an die V24-RXD, TXD Leitungen am SubD-Anschluß oder intern am Kartenleser betrieben werden. Ein Loopback durch den EP200 ist möglich (wird für die Kartenkommunikation auch benötigt) und wird zum Test der Schnittstelle verwendet (SC5BOS).

BF-Bus

Der BF-Bus ist ein serieller Eindraht-Bus, der den Telefonhörer und die Optionsbaugruppen mit dem C5-Grundgerät verbindet. Die Kommunikation erfolgt dabei paketbasiert, halb-duplex und bidirectional mit 19200baud 8N2 und TTL-Pegel (der inaktive Zustand ist High). Physikalisch ist der BF-Bus fest mit dem Seriellen Port 0 des NEC verbunden. Zur Kommunikation benötigt SC5BOS alle Ser0-Interrupt-Vekoren und zusätzlich den Timer0 im SingleShot-Mode.
Die Kollisionsvermeidung (da mehrere sendefähige Baugruppen) erfolgt durch einen einfachen Signalisierungsmechanismus: Die sendewillige Station signalisiert Ihren bevorstehende Bus-Benutzung mit dem Umschalten der BF-Bus-Leitung auf den aktiven Zustand (Low) für ca. 3-5ms. Alle anderen Baugruppen erkennen dies und halten die BF-Bus-Leitung ebenfalls auf Low. Die sendende Baugruppe (z.B. Höher) kann nach Umschalten auf den inaktiven Zustand erkennen, ob die Leitung weiterhin auf Low bleibt. Nachdem auch die anderen Baugruppen die Leitung wieder inaktiv geschaltet haben, beginnt die sendewillige Baugruppe (nach ca. 3-13ms) mit der Aussendung des (ersten) Paketes. Folgen noch weitere Pakete, so können diese in einem Zeitfenster auch ohne Bus-Request hinterhergesendet werden. Nach ca. 5ms erfolgt bei Bedarf vom Empfänger ein Antwort(ACK)-Paket. Das Zeitfenster für weitere Pakete beträgt ca. 10ms.

Jedes Paket ist nach folgenden Schema aufgebaut:

Absenderadresse	Empfängeradresse	Länge Paket	Nutzdaten (Länge-4 Bytes)	XOR Prüfsumme

Folgende Nutzdaten kann der Hörer zum Grundgerät schicken:

- **2Eh, 01h, Key** (Key enthält den Tastencode, beim Loslassen der Taste +80h)
- **2Fh, S01, S02, S03, S04** (S0x = Antwort auf 0Fh, enthält u.a. Status den Reed-Kontakts)

Die Nutzdaten, die das Grundgerät zum Hörer schicken kann, sind wie folgt aufgebaut:
Das erste Datenbyte ist ein Kommando. Danach folgen mit variabler Länge die Parameter zu einem Kommando.
Folgende Kommandos werden vom Hörer unterstützt:

- **02h** - Textausgabe, es werden die folgenden 16 ASCII-Zeichen im Display dargestellt (s. Zeichentab.)
- **06h** - Symbolanzeige, mit 7 Steuer-Bytes werden die Symbole im Display ein, aus oder blinkend geschaltet
- **07h** - Einstellung, mit einem Steuerbyte werden Licht, Hörkapsel aus/ein, Mikrofon-Auswahl und die Hörerlautstärke gesteuert
- **08h** - Töne/Klingeln ausgeben, durch 6 Bytes wird eine Tonsequenz über den „Piepser“ ausgegeben.
- **0Fh** - Version/Statusabfrage, keine näheren Infos **L**
- **10h** - Bestätigung (ACK) senden, keine Parameter

Typ 06h: Display-Symbole

Bsp	Beschreibung	Details (Bits)
00h	Absender	
1Ah	Empfänger (Hörer)	
0Ch	Länge 12 Bytes	
06h	Typ 6 Paket	
00h	Display-Symbole	1=Schlüssel, 2=Pfeil heraus, 4=Pfeil hinein, 8=Lautsprecher, 16=Hörer, 32=Kreis
01h	Display-Balken	1=S-Rahmen, 2=S1, 4=S2, 8=S3, 16=S4, 32=S5, 64=TAE-Rahmen, 128=TAE-Symbol
00h	???	
20h	Display-Symbole blinkend	1=Schlüssel, 2=Pfeil heraus, 4=Pfeil hinein, 8=Lautsprecher, 16=Hörer, 32=Kreis, Rest=???
01h	Display-Balken blinkend	1=S-Rahmen, 2=S1, 4=S2, 8=S3, 16=S4, 32=S5, 64=TAE-Rahmen, 128=TAE-Symbol
00h	???	
00h	???	1=komplettes Display blinkt
30h	Prüfsumme	

Typ 07h: Einstellungen

Bsp	Beschreibung	Details (Bits)
00h	Absender	
1Ah	Empfänger (Hörer)	
06h	Länge 6 Bytes	
07h	Typ 7 Paket	
71h	Einstellungen am Hörer	1=Licht ein, 2=Hörkapsel ein, 4=Mikro ein, 8=Auswahl Freisprech-Mikro, 16, 32, 64=Hörerlautstärke (8 Stufen)
6Ah	Prüfsumme	

Typ 08h: Tonerzeugung

Bsp	Beschreibung	Details (Bits)
00h	Absender	
1Ah	Empfänger (Hörer)	
0Bh	Länge 11 Bytes	
08h	Typ 8 Paket	
C1h	Tonhöhe 1	Bei Tonhöhe2=0x4F ist Periodendauer des Tons T=(256-Tonhöhe1)*8,138µs
4Fh	Tonhöhe 2	???
0Fh	Tonlänge	

01h	Pausenlänge	
01h	Tonanzahl	
13h	Steuerbits	1=Ton ein, 2=Single shot, 4=???, 8=Portamento, 16=Tastenklick, 32=leiser Beep
8Bh	Prüfsumme	

Typ 2Eh: Tastencode

Bsp	Beschreibung	Details (Bits)
1Ah	Absender BT	
00h	Empfänger GG	
07h	Länge 7 Bytes	
2Eh	Typ 2Eh Paket	
01h	???	
30h	Tastencode	Beim Loslassen einer Taste wird der Tastencode plus 80h gesendet.
02h	Prüfsumme	

Die (Tasten-)Nachricht muss vom Grundgerät mit dem ACK-Paket (Typ 10h) bestätigt werden, ansonsten wird sie automatisch nach einigen Millisekunden wiederholt.

Kartenleser

Der Kartenleser ist mit CC_Clock und CC_Data an den EP200 angeschlossen. Wahrscheinlich erzeugt der EP200 analog zum I²C-Bus das Clocksignal automatisch, wenn Daten gesendet oder ausgelesen werden. Die Kommunikation mit der Karte erfolgt dabei in der Originalsoftware mit 9600baud über Ser1. Daher ist auch CC_Data physikalisch mit RXD1 verbunden.
Weitere Nachforschungen werden später noch vorgenommen. Ziel: Einfache Benutzung einer Standart-Speicherkarte (E²Prom) als transportable und am PC editierbare Frequenzspeicherablage.

externe Schnittstelle

Die externen Leitungen V24-RXD und V24-TXD können durch den EP200 mit Ser1 verbunden werden. Dadurch ist es möglich ohne Umbau bei geschlossenem Gerät Daten zu transportieren. In der Originalsoftware wurden im Servicebetrieb Meldungen mit 19200baud ausgegeben. SC5BOS nutzt diese Schnittstelle jedoch viel intensiver und stellt eine PCP-Kommunikation mit bis zu 57600baud bereit. Eine höhere Standartbaudrate (115k2) ist aufgrund der Taktfrequenz von 4.032MHz des NEC nicht machbar, da es dafür keine ganzen Teiler mehr gibt. Die V24-Leitungen haben jedoch einen entschiedenen Nachteil: Sie führen nur ein TTL-Signal (5V – 0V). Eine PC-RS232-Schnittstelle benötigt jedoch ein Signal mit rund ±10V. Daher ist ein einfacher Pegelwandler wie z.B. ein Max232A oder Max202 (viele Typen) nötig. Die Flusskontrollsignale V24-CTS, V24-RTS werden nicht benutzt, um keine zusätzliche Konkurrenz für den I²C-Bus (Signale gleich I²C-Clock und -Data) darzustellen.

I²C-Bus und Slave-IC's

Der I²C-Bus wird komplett durch den EP200 gesteuert. Er stellt 2 byte-große Register die an den Adressen 0E002h und 0E003h eingeblendet sind (MemoryMapped-I/O).
An 0E002h werden die Datenbytes des I²C-Busses geschrieben (oder ausgelesen), während an 0E003h Controlbits die Übertragung steuern und einen Status der Übertragung beinhalten.
Die I²C-Bus-Benutzung funktionier jedoch erst, nachdem der EP200 initialisiert worden ist. Welches Controlbit dabei die Clocksignalgenerierung für die Ausgabe einschaltet liegt leider noch im Dunkeln. SC5BOS initialisiert den EP200 nach dem Reset (so wie Originalsoftware) so, das I²C funktioniert. Ist I²C erst mal aktiv, wird mit jedem Schreibzugriff auf das Datenregister das geschriebene Byte (+ 9tes Bit) auf den Bus geshiftet. Das 9te Byte stellt bei I²C-Übertragungen das ACK-Bit dar. Weitere Infos zu I²C bitte aus Datenblättern o. ä. herausnehmen.

I²C-Controlregister

Damit eine korrekte Übertragung getätigt werden kann, muss der EP200 verschiedene Conditions auf dem I²C-Bus erzeugen können. Es handelt sich dabei um die im I²C-Protokoll spezifizierten Start- und Stop-Conditions. Auch fehlt im Datenregister noch das 9te Bit (Acknowledge). Daher existieren im Controlregister folgende Bits:

Bit0: "1" erzeuge Start-Condition (SDA H->L, bei SCL = H).

"0" keine Start-Condition (SDA, SCL werden beim Start einen Clock-Zyklus auf LOW gesetzt).
Bit1: "1" erzeuge Stop-Condition (SDA L->H, bei SCL = H).
"0" keine Stop-Condition (SDA, SCL werden nach dem 9ten Bit auf LOW gesetzt).
Bit2: "1" ACK = HIGH : Nicht bestätigen oder Slave bestätigt.
"0" ACK = LOW : Slave bestätigen (beim Daten-Einlesen)

Damit die Programmroutine weiß, wann ein ins Datenregister geschriebenes Byte vollständig herausgeschiftet ist, existier im Controlregister ein Busy-Flag (Bit 3), das solange HIGH bleibt, wie der EP200 das Datum herausshiftet. Das Bit 2 (Acknowledge) scheint beim Auslesen den Zustande der Datenleitung SDA zu beinhalten. Damit kann detektiert werden, ob ein Slave das Datum bestätigt hat.

Folgende Bitkombinationen für das Controlregister sind üblich:
TX:
05 101 Erstes Zeichen an Slave, Slave bestätigt mit ACK->L
04 100 ein weiteres Zeichen wird gesendet,
06 110 Das letzte Zeichen wird gesendet, Slave bestätigt mit ACK->L oder I²C-Reset
RX:
00 000 Ein Zeichen wird empfangen und vom Master bestätigt
06 110 Das letzte Zeichen wird vom Master empfangen und NICHT bestätigt

I²C-Timing

Die Pulslänge eines Clock-Signals auf der SCL-Leitung liegt bei 15,4µs (High =7,5µs, Werte gemessen). Die Zeit von der ersten steigenden Flanke bis zu letzten entspricht rund 123,2µs. Eine Ausgabe eines Bytes mit Start- und Stop-Condition beträgt rund 162,4µs. Das Timing ist somit für normale I²C-IC's völlig unkritisch und liegt bei ca. der halben Frequenz, die als obere Grenze bei den in C5 eingesetzten IC's spezifiziert ist.

I²C-Bausteine im C5

Im C5 sind 3 Bausteine mit über den I²C-Bus an den EP200 angeschlossen:

- die PLL („TBB200" od. „Zarlink NJ88C33")
- der kombinierte AD/DA-Wandler PCF8591 von Philips
- ein E²Prom 24C02A mit 256Byte Speicher

Mit der PLL und dem AD/DA-Wandler klappt die Kommunikation hervorragend. Mit dem E²Prom ist noch nicht so viel anzufangen, hier muss noch einmal das s.g. „Acknowledge-Polling" überprüft werden – erste Test waren lieferten recht merkwürdige Ergebnisse (nur 4 statt max. 16Byte beim PageWrite geschafft etc.).

PLL
Die PLL arbeitet im double Modulus Betrieb mit einem Vorteiler-IC mit Frequnez/64 und Frequenz/65. Die Referenzfrequenzquelle liegt im Empfänger des C5, liefert 14,85MHz und ist über den PCF8591 justierbar. Um ein Raster von 12,5kHz zu bekommen, wird der Referenzteiler (RCounter) auf 1188 gesetzt. Der Gesamtteiler des Systems muss jetzt so gesetzt werden, das die gewünschte Oszillatorfrequenz / Gesamtteiler = 12,5kHz ergibt. Dieser Teiler (Counter) muss nun so „zerlegt" werden, das A- und N-Counter entstehen. Es gibt mehrere Möglichkeiten dafür. die einfachste: Der N-Counter = Counter DIV 64, falls Counter MOD 64 > 0, sonst N-Counter = Counter DIV 64 – 1. Der A-Counter = Counter MOD 64, falls Counter MOD 64 > 0, sonst A-Counter = 64.
Natürlich muss die PLL auch initialisiert werden (Status, im C5 = 7Eh). Einfach Datenblatt einsehen.

AD/DA-Wandler
Der Wandler ist weiter nichts Besonderes: Ein Lesezugriff befördert ein Byte aus dem internen Datenregister des Wandlers nach Draußen. Mit Schreibzugriffe können der Wandlungsmodus, die Kanalverschaltungen, der als nächstes zu wandelnde Kanal oder der Ausgabewert verändert werden. Einfach Datenblatt einsehen.

EEProm 24C02A
Das EEProm im C5 erhält eine Sonderbehandlung. Während die Schreiboperation mit max. 2 Bytes wie im Datenblatt beschrieben arbeitet, funktioniert das sequentielle Lesen im C5 offensichtlich nicht. Daher wird für jedes Byte eine Random-Read-Operation ausgeführt. Dies ist auch der Grund, warum extra API-Funktionen zum Zugriff bereitstehen. Der Programmierer braucht sich keine Gedanken um die o.g. Unzulänglichkeiten zu machen.

Motorola DSP 56001 / 56002

Der Motorola DSP 56001 ist ein inzwischen veralteter „klassischer“ digitaler Signalprozessor mit 24bit-Festkomma-Arithmetik. Er verfügt über 512Wörter Programmspeicher und 2 mal 256Wörter Datenspeicher (X, Y). Auch hat er eine 256Wörter große Sinustabelle mit an Board. (Jedes Word hat natürlich 24Bit.)
Externen Speicher oder ROM wurde im C5 nicht angeschlossen. Programme und Daten bekommt der DSP über das Host-Interface (HI), einem parallelem 8bit breiten High-Speed-Port, der mit dem EP200 verbunden ist. Der NEC kann den HI über eine Speichereinblendung erreichen und erhält zusätzlich noch ein INT-Signal vom DSP. Seine eigentlichen Daten bekommt der DSP über seine beiden SPI-Schnittstellen an denen je ein Audio-PCM-Codec angeschlossen ist. Die Taktung des SPI-Busses und der Codecs übernimmt dabei der EP200, im DSP sind nur die beiden RX-Ints zu behandeln.

Host-Interface

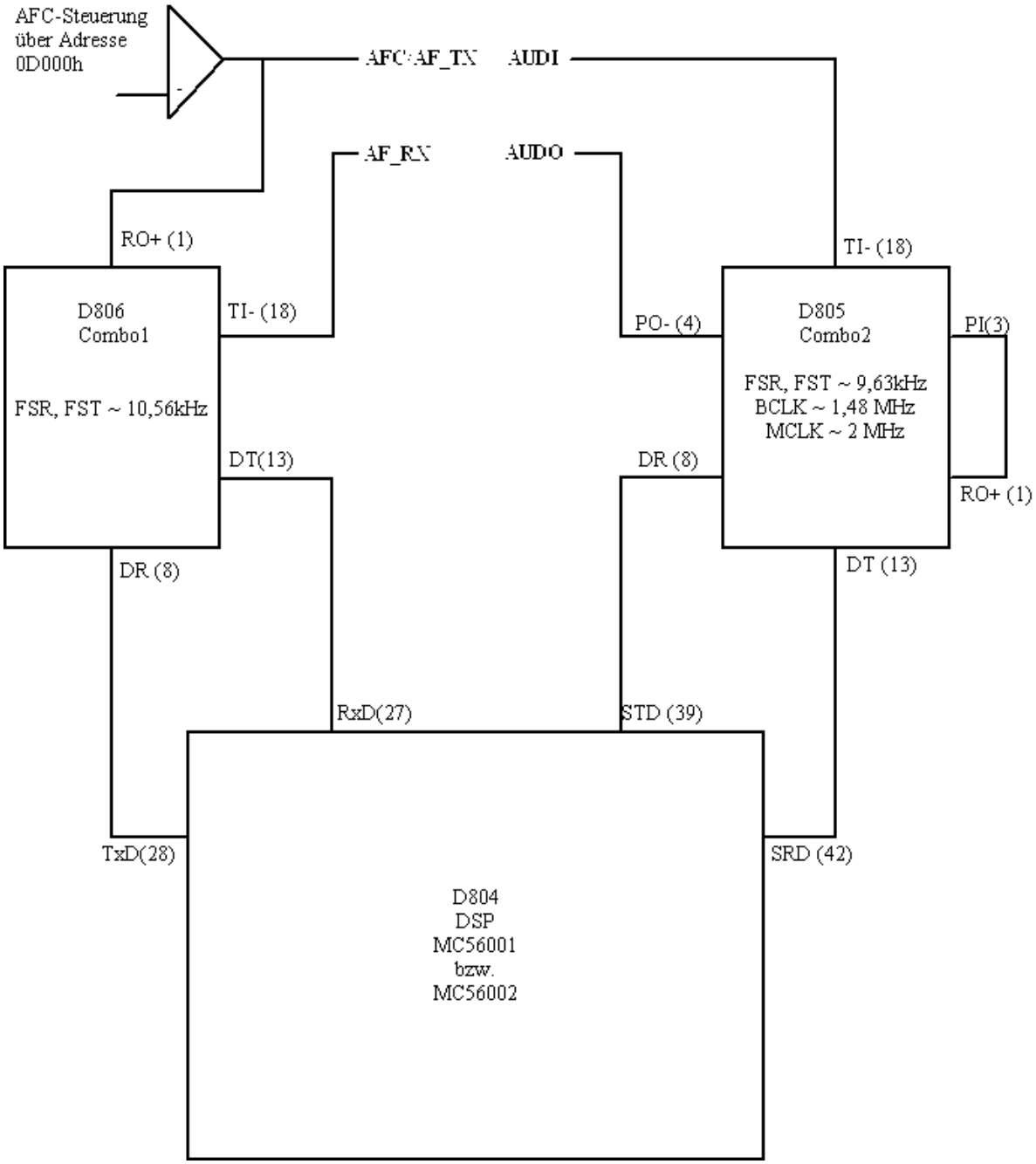
(in Arbeit)
Weitere Informationen zu diesem Thema findet man im UsersManual der DSP56000-Familie.

SCI/SSI-Ports

(in Arbeit)
Weitere Informationen zu diesem Thema findet man im UsersManual der DSP56000-Familie.

Der Codec-Signalweg

Aufgrund der kombinierten analogen Sprach- und digitalen Datenübertragung ergeben sich Unterschiede zwischen dem AF-Codec-Teil (an Modulator / Empfänger angeschlossen) und dem NF-Codec-Teil. Die beiden an sich gleichen MC145480-PCM-Codecs werden mit unterschiedlichen Samplingraten betrieben (siehe Bild). Damit wurde die Sprache einfach durch Pufferung im DSP komprimiert. Weiterhin hat der DSP die Sprachverschleierung (Invertierung) vorgenommen und sicher auch DTMF-Töne erzeugt.



Zeichensatz des Standard-Hörers

Der Hörer des C5 verfügt nicht über einen kompletten ASCII- oder erweiterten ASCII-Zeichensatz. Es können daher nur folgende Zeichen (20h bis 91h) dargestellt werden:

	2xh	3xh	4xh	5xh	6xh	7xh	8xh	9xh
x0h		0	@	P	`	p	Ä	¿
x1h	!	1	A	Q	a	q	Ö	..
x2h	"	2	B	R	b	r	Ü	
x3h	#	3	C	S	c	s	ä	
x4h	$	4	D	T	d	t	ö	
x5h	%	5	E	U	e	u	ü	
x5h	&	6	F	V	f	v	ß	
x7h	'	7	G	W	g	w	®	
x8h	(	8	H	X	h	x	¬	
x9h	)	9	I	Y	i	y	-	
xAh	*	:	J	Z	j	z	¯	
xBh	+	;	K	[	k	{	8	
xCh	,	<	L	\	l	|	►	
xDh	-	=	M	]	m	}	◄	
xEh	.	>	N	^	n	~	▲	
xFh	/	?	O	_	o	▉	▼	

In wieweit eigene Zeichendefinitionen in den Hörer geladen werden können steht nicht fest. Wahrscheinlich geht so etwas nicht. Zeichen mit Nummern unter 20h (0 bis 31) werden vom Hörer ignoriert – der Rest des Textes rückt einfach nach. Alle Zeichen mit Nummer größer 91h sind ebenfalls '**..**' (horizontaler Doppelpunkt). Das Zeichen 8Bh stellt das Netzstecker-Symbol dar. das Zeichen 90h ist ein solcher Return, der Pfeil beginnt jedoch von unten (an x-Achse gespiegelt).

Tastaturcode des Standard-Hörers

Taste	Code	ASCII	Taste	Code	ASCII
0	30h	'0'	*	2Ah	'*'
1	31h	'1'	#	23h	'#'
2	32h	'2'	C	3Ah	':'
3	33h	'3'	.	3Bh	';'
4	34h	'4'	..	3Ch	'<'
5	35h	'5'	◄	4Bh	'K'
6	36h	'6'	►	21h	'!'
7	37h	'7'	▲	3Fh	'?'
8	38h	'8'	▼	3Dh	'='
9	39h	'9'	(	3Eh	'>'
			Kontakt	48h	'H'

SC5BOS erzeugt für die ¤-Taste (On/Off) zusätzlich noch den Tastencode **27h** beim Loslassen dieser Taste (natürlich nur wenn sie nicht zu lange gedrückt wurde und das Telefon abschaltet). Jedoch gibt es kein BF-Bus-Paket dafür, der Tastencode wird nur dem Tastaturhandler übergeben.

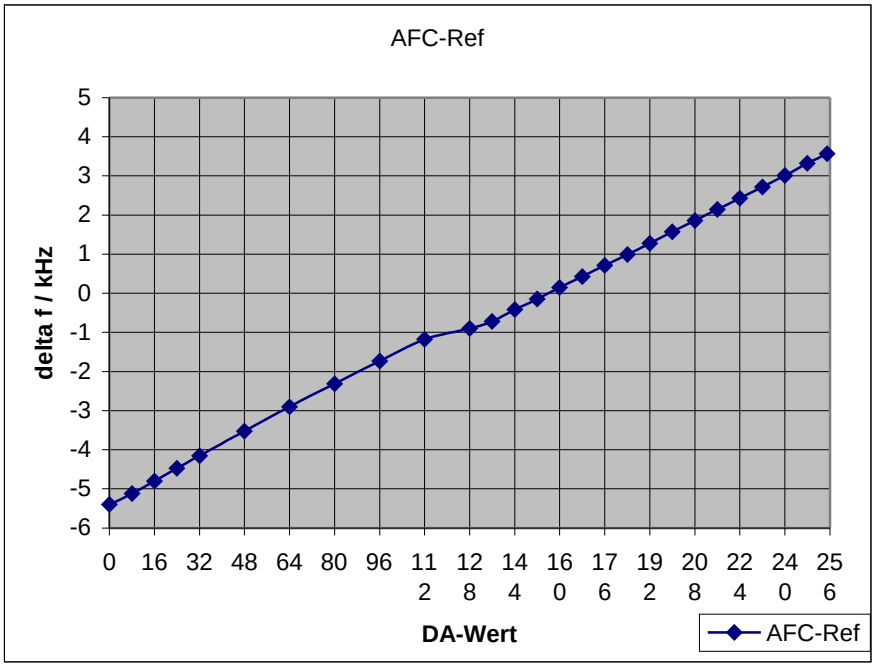
Der Empfänger

Der Empfänger besitzt leider keine so ausgezeichnete Empfindlichkeit wie moderne 70cm-Geräte. Direkt (am Eingang, ohne Duplexer oder Koppelschaltung) gemessen liegt sie bei -114,5dbm (20db SINAD). Die Werksangabe von Siemens/ABB gibt –113dbm an.

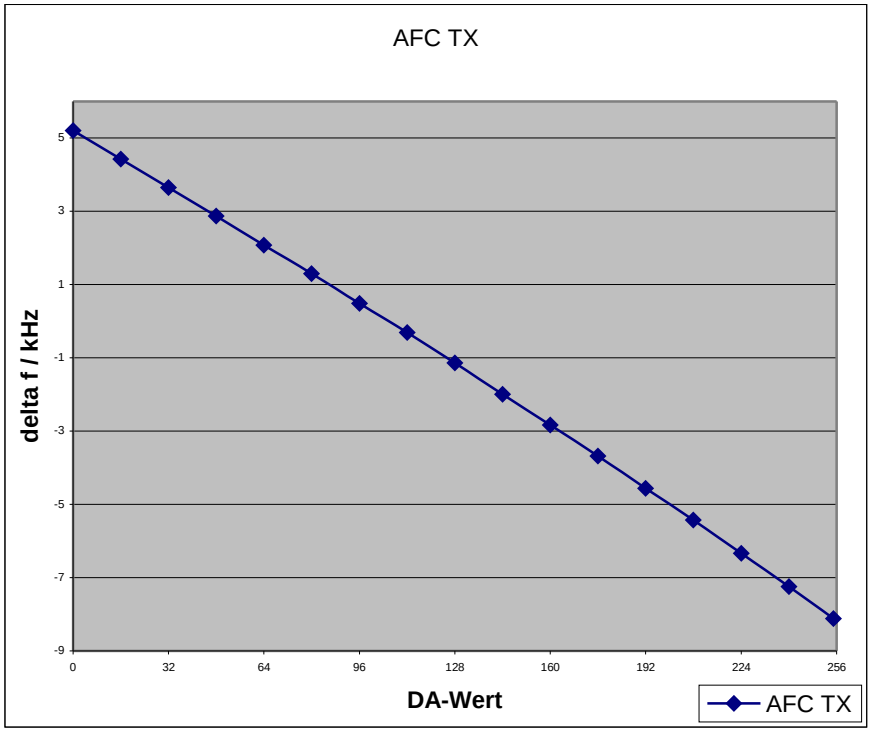
Die AFC Steuerung

Die PLL bekommt ihre Referenzfrequenz aus dem Empfänger. Hier sitzt ein silberner Block, der eine Frequenz von 14,85 MHz produziert. Dieser ist über eine Verstellschraube abgleichbar. Neben dieser Möglichkeit das

Gerät abzugleichen, gibt es noch 2 per Software veränderliche Größen, als AFC_Ref und AFC bezeichnet. Der erste Wert wird über die I²C-AD/DA-Wandler eingestellt und stimmt die Referenzquelle (14,85MHz) fein ab. Der Regelbereich liegt dabei bei etwa ± 4,5 kHz der Oszillatorfrequenz. Pro Digit wird die Frequenz also um rund 35Hz verändert (pro 16 Schritte ~ 560Hz).



Die andere Einstell-Möglichkeit ist AFC per parallelem DA-Wandler D807. AFC gleicht die evtl. auftretenden Versatz der Sendefrequenz zur Empfangsfrequenz aus, indem dem NF-Signal (AF_TX) ein Gleichspannungsoffset überlagert wird. Der Einstellbereich entspricht ca. ± 6,6 kHz der Sendefrequenz. Pro Digit wird die Sendefrequenz um ca. 52 Hz verstellt (ca. 833Hz pro 16Schritten). Eine Erhöhung des Wertes bewirkt jedoch eine Verringerung der Sendefrequenz.



Paket Communication Protocol

Zur Kommunikation mit dem Telefon wird das dafür entwickelte einfache Protokoll PCP eingesetzt. Dieses Protokoll findet Anwendung bei sämtlichen BOSs und bei einigen anderen kleinen AVR-Projekten. Durch diese Mehrfachverwendung kann die PC-Software mehrfach verwendet werden. Das Protokoll bietet nur Übertragungssicherheit auf Paketebene, jedoch keine Fehlerkorrektur wie z.B. AX25. Das jeweilige Anwendungsprogramm muß daher selbst eine Behandlung für fehlende (fehlerhaft übertragene) Pakete implementieren.

Art und Nutzung

Das Packet Communication Protocol (im folgenden PCP genannt) wurde entwickelt, um eine standardisierte Schnittstelle zu einem µContoller-basierten Gerät zur Verfügung zu haben und damit die Verwendung von einheitlichen (Hilfs-)Programmen zu ermöglichen. Die Hauptaufgaben dieser Programme sind u.a. Protokollieren, Testen und Modifizieren von Speicher. Gerätespezifische Programme dagegen meist können parallel benutzt werden, und benötigen keine Protokoll- oder Kommunikationsimplementierung mehr. Von der Art her ist das PCP nach OSI-Schichtenmodell ein Layer2-Protokoll, welches einen Paket-orientierten Versand und Empfang von Daten ermöglicht und mit einer Prüfsumme korrekte Daten gewährleistet. Eine Verbindungs- oder Flusskontrolle ist kein Bestandteil von PCP. Fehlerhafte Pakete werden verworfen.

Rahmenaufbau

Das PCP definiert 2 Rahmentypen. Es wird zwischen einem Minimal-Rahmen (Short Frame) für ein einzelnes Daten-Byte und einem großen (Long Frame) Rahmen für einen Datenblock unterschieden. Die Art des Rahmens wird in den oberen 4 Bits des zuerst gesendeten Zeichens mitgeteilt. Enthalten die Bits nur Einsen (0Fh), so handelt es sich um einen Long-Frame. Ist jedoch nur das unterste Bit = „1“ (01h), so handelt es sich um einen Short-Frame. Alle anderen Bitmuster sind für die oberen 4Bits des Startzeichens ungültig. Die unteren 4 Bits dieses ersten Zeichens enthalten die Zielportnummer. Es sind nur die Portadressen „0“ bis „9“ zulässig. Die Adressen „A“ bis „F“ sind ungültig und werden ignoriert.

Short Frame

Nach dem Startzeichen (1xh, x steht für den Port 0..9) folgt genau ein Datenbyte (Es werden immer die kompletten 8bit übertragen). Als drittes und letztes Zeichen folgt eine 8bit-lange Prüfsumme (CRC), die nach dem XOR-Verfahren aus Startzeichen und Datenbyte berechnet wurde. Pakete mit fehlerhafter CRC werden vom Empfänger verworfen.

Beispiel:

Start	Data	CRC
10h	41h	51h

Eine Übertragung größerer Daten mit Short-Frames ist nicht zu empfehlen, da die effektive Bandbreite dabei gedrittelt werden würde (Beispiel: Seriell mit 115200baud **à** 11,5Byts/s. Bei SF: ~ 3,8Byte/s Nutzdaten). Gedacht ist dieser Rahmentyp für parameterlose Kommandos und Bestätigungen (ACK, NAK).

Long Frame

Beim Long Frame folgt nach dem Startzeichen (0Fxh, x steht für den Port 0..9) das Längenbyte. Dieses Byte enthält die Länge der folgenden Nutzdaten **minus eins**. Da die Länge um eins verringert wurde, können volle 256 Byte Nutzdaten übertragen werden (Längenbyte enthält 255). Jedoch ist das Versenden eines leeren Paketes so nicht mehr möglich (und auch nicht erwünscht). Die minimale Anzahl der Datenbytes beträgt Eins (Längenbyte enthält 0). Dies ist ein Pluspunkt des PCPs, das so auch andere einfache Protokolle direkt kapseln kann, deren Längenangabe sich auf die komplette Paketlänge bezieht. Nach dem Längenbyte folgen nun genau Längenbyte+1 Bytes Nutzdaten, die wie beim Short Frame beliebig sein können.
Im Gegensatz zum Short Frame wird der Long Frame mit einer 16bit Prüfsumme (CRC nach CCITT) gesichert. Sie folgt direkt nach den Nutzdaten, wobei das höherwertige Byte der CRC zuerst übertragen wird.

Beispiel:

Start	Länge	Data0	Data1	Data2	CRCh	CRCl
F0h	02h	41	61h	62h	xxh	xxh

Bei voller Ausnutzung des Long Frame werden mit 4-Rahmen-Bytes 256 Nutzbytes übertragen, was rund 98% der Bandbreite entspricht.

CRC's

Die Prüfsummen dienen der Übertragungssicherheit und werden durch die PCP-Implementierung erzeugt. Beim Empfang wird die generierte Prüfsumme des Datenteils (komplettes restliches Paket) mit der empfangenen Prüfsumme verglichen – stimmen diese nicht überein, so wird das Paket verworfen (fehlerhaft). Die Applikation muß in diesem Fall selbst für eine erneute Übertragung sorgen.

XOR-CRC

Für die Short-Frame-Pakete wird nur eine 8bit-Prüfsumme verwendet, die sich durch folgende Gleichung darstellt: CRC = Start XOR Data.

CRC nach CCITT

Der Bildungsalgorithmus dieser genormten CRC-Funktion ist recht kompliziert, jedoch sehr sicher. Er ist in entsprechender Fachliteratur zu finden. Im PCP-Server und dem SC5BOS wird der Schnelligkeit wegen eine vorgerechnete Tabelle (256 Words) verwendet. In den AVR-Projekten wird der Algorithmus komplett berechnet (Assembler) da hier der RISC-Prozessor schnell genug, jedoch Speicherplatz für die Tabelle kaum vorhanden ist.

Ports (Kanäle)

PCP unterscheidet bei der Kommunikation 10 unterschiedliche Ziel-Ports (virtuelle Kanäle). Eine Anwendung sollte immer nur einen Kanal sowohl fürs Senden, wie zum Empfangen benutzen. Die ersten 3 Ports sind in Ihrer Bedeutung und in der Kommunikationsart jedoch festgelegt und bilden einen Standart, der allgemeine Funktionalitäten geräteunabhängig definiert.

Master – Slave – Beziehungen

Die Kommunikation mit einem µController-basierten Gerät erfolgt meist in einer Master-Slave-Beziehung der folgenden Art: Der Master (PC) sendet ein Kommando (Request) und erwartet eine Antwort (Response) vom Gerät, deren Nutzdaten abhängig vom Request interpretiert werden. Dieses Verhalten ist jedoch nicht Bestandteil vom PCP, sondern wird von der Applikation, die PCP benutzt, aufgesetzt.
Jedes Request mßs mit einer Response bestätigt werden. Eine Response jedoch bedarf keiner Bestätigung. Enthält eine Response keine spezifischen Daten, so wird ein ACK-Zeichen für „erfolgreich“ oder ein NAK-Zeichen für „nicht möglich“ zurückgesendet. Die Zeichen sind ASCII-Tabellenkonform (ACK = 06h, NAK = 15h). Requests dürfen als erstes Zeichen **kein** ACK oder NAK enthalten.
ACK oder NAK ist jedoch nur eine Möglichkeit eine Response zu generieren. Es können beliebe antworten geschickt werden.