digisolutions

PP5BOS

Philips Porty 5x Basic Operating System

Version 0.1d

Ein Projekt von
DF2KW, Guido Küppers
DO1FJN, Jan Alte

Inhalt:

Terminologie im Dokument

- Bootloader Teil des PP5BOS der mit dem Benutzer interagiert.
- Porty Bezeichnet das Telefon Philips Porty FG52 / FG53
- PP5BOS Philips Porty 5x Basic Operation System, dieses
- PP5PRG Philips Porty 5x Program, Programmdatei die in einem Porty mit PP5BOS funktioniert.
- ZST Zentralsteuerung, der V40-Controller
- BIF Bus Interface
- BST Bussteuerung, IC zum Bedienen des I²C-Busses
- DST Datensteuerung
- MST Messsteuerung

Einleitung

warum eine Umbauanleitung?

Ziele

Bei diesem Projekt geht es vorrangig darum, kostengünstig und mit minimalem Aufwand das Porty Telefon in einen FM-Amateurfunktranceiver für das 70cm Band (430-440 MHz) zu verwandeln.

Beschreibung des Projektes

Das Projekt „PP5BOS“ beschäftigt sich mit der Softwareentwicklung für das alte C-Netztelefon Philips Porty FG52 und FG53. Dieses Telefon verfügt über einen NEC V40 Microcontroller (µC). Dieser ist softwarekompatibel zu einem i80186 und besitzt zusätzlich noch erweiterte Befehle. Er ist ansonsten ein klassischer Controller (keine CPU) mit UART, Ports, etc.). Software kann deshalb mit „alten“ PC-Assemblern und Compilern erstellt werden.
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 simples 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 Sysntaxhighlighting 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 Porty mit dem PC zu Verbinden, wurde ein serieller Pegelwandler (TTL←→RS232) benötigt. Als einfache Lösung wurde ein Max202 und ein kleiner 5V-Stabi auf einer Lochrasterplatine zusammen mit 2 Pfostensteckern gelötet. Die Beschaltung entspricht der im Max202-Datenblatt. Angeschlossen wird diese Baugruppe an die 15pol. Buchse des Porty. Über diese Buchse erfolgt auch gleich die externe Stromversorgung. Folgende Pins müssen mit dem Max202 verbunden werden: ---
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 Porty gelangen. Als Flash verwendet der Autor den DIL-Typ 29F010 der Firmen AMD, TI und Hyundai. Alternativ einsetzbar sind auch die Typen 29F002TC / BC bzw. 29F040. In jedem Fall muss das Original-ROM aus dem Porty entfernt und der Flashrom eingesetzt werden.

Software:
Die Software PP5BOS ist nicht das einige was im Laufe der Entwicklungsarbeit entstand. Einige Features in Kurzform:

- das Porty könnte komplett per PC gesteuert werden
- neue Software wird einfach bei geschlossenem Gerät ins Porty übertragen (PP5BOS ist so updatefähig)
- mehrere Dienste können parallel laufen (z.B. PacketRadio + Debugger + Bediensoftware)
- Entwickler brauchen sich nur noch um die eigentliche Aufgabe des Programms kümmern, den Rest erledigt dann PP5BOS (I/O, Tastenabfrage etc.)

Unweigerlich entstand auch PC (Windows-) software: Diese unter Delphi5 geschriebenen Tools sind für die Kommunikation mit dem Telefon (oder „kompatiblen" Geräten) gedacht. Kern der Kommunikation ist ein als PCP (Packet Communication Protocol) bezeichnetes Übertragungsprotokoll. Es spezifiziert eine auf Paketen aufgebaute Kommunikation zwischen 2 Geräten (hier Porty und PC). Eine nähere Beschreibung folgt später. Die Fähigkeiten der Windows-Software in Kurzform:

- OLE-Server für die Kommunikation (PCP) über eine serielle (Standart-) Schnittstelle-
- Einsehen vom RAM und ROM per PCPMemoryViewer
- Ausgabe von Textmeldungen des Porty per PCPMessageViewer
- Übertragen und ausführen von Programmen im per Telefonprogrammierer
- Beschreiben des Flash-ROM per Telefonprogrammierer

(Diese Software wird in einem extra Dokument behandelt.)
Der eigentlicher Kern der Entwicklung – PP5BOS – wird im Folgenden beschrieben.

Lastenheft PP5BOS

- das PP5BOS soll in einem 16Kbyte großen geschützten Sektor des Flash-Roms Platz finden
- es initialisiert die nötige Hardware, die für eine Programmierung des Flash-Bausteins nötig ist
  dazu zählen:
  - Prozessorregister
  - externe serielle Schnittstelle (9600 oder 19200baud, 8N1)
  - Interrupt-Vektortabelle
- es soll ein einfaches Kommunikationsprotokoll implementiert sein, das eine sichere Datenübertragung gewährleistet. Daher ist eine CRC16 (nach CCITT) wünschenswert.
- Der Telefonhörer soll bereits Meldungen ausgeben können
- Über die Tastatur des Hörers sind im Bootloader Informationen abrufbar und Testfunktionen ausführbar.
  darunter fallen:
  - Versionsausgabe
  - CRC nach CCITT des Flashbausteins
  - Flash löschen (bis auf PP5BOS)
  - Einstellen der Übertragungsparameter, abweichend vom Standart (z.B. 9600baud).
  - Speichertest
- der Funktionsumfang ist einfach zu erweitern
- Über das Kommunikationsprotokoll kann ein Programm in den Speicher geladen und ausgeführt werden.
- Über das Kommunikationsprotokoll kann ein Programm in den Flash geschrieben werden.
- Über das Kommunikationsprotokoll kann ein beliebiger Datenblock in den Flash geschrieben werden.
- Über ein API werden Basisfunktionen den Anwendungsprogrammen zugänglich gemacht.
- Eintreffende Daten sind über eine Event-Schnittstelle dem laufenden Programm zu übergeben.
- ...

Ideensammlung für Kreative

Hardware

- Als Programmspeicher sollte ein Flash eingesetzt werden, das ein Bootloader enthält. Ein Update der Software ist dann über eine serielle Verbindung zum Porty möglich.
- Statt Hörer Anschluss von Bedienteilen
  - Display 4x20 Zeichen oder Grafikpanel (320x240 dots)
  - Impulsdrehgeber für Volume – Frequenz (Umschaltung durch Drücken, Timeout)
  - 10er Tastatur mit aufgedruckten Buchstaben (wie „Handys“)
  - Buchstabeneingabe über Drehgeber, Tasten
  - 4 Softkeys am Display-Rand
  - Mikrofonanschluss 2,5 Klinke
  - NF-Out oder kleine NF-PA für 4Ohm-Lautsprecher
- Zusatzanschlüsse auf wechselbarer Platine, von Software Erkennbar (Adresse etc.)
- Kopplung eines anderen Funkgerätes 2m, CB etc. statt Hörer: Relaisbetrieb!

Software

- Beibehaltung der Digital-Platine rund um den NEC V40
- FM-Empfang 12,5 / 25 kHz – Raster
- HF-Ausgangsleistung veränderbar
- Erzeugen von DTMF-Tönen und dem 1750Hz Rufton
- Kartenleser für (Krankenkassen-)Speicherkarten nutzbar
- Ablage von Parametern auf o.g. Karten
- Relais und DigipeaterListe (Updatebar) von EU/DL
  → durch Eingabe das Locators sind 70cm Relais auswählbar (im Umkreis erreichbare)

Umbau der Steuerung

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. Modifikation POWON-Signal: 1 Stück Litze, eine Leiterbahnunterbrechung, 1 Stück Fädeldraht.
2. Modifikation VSDB-Signal: 1 Stück Fädeldraht
3. **Nur bei 29F040 erforderlich:** Auftrennen der Verbindung von Pin1 (A18) und Pin32(Vcc).
4. **Nur bei 29F040 erforderlich:** Verbinden des PLCC-Pin1 mit der A18 des µC
5. Entfernen des Pull-Up-Widerstands des WE (Pin31).
6. Verbinden des WE (Pin31) mit dem WE am SRAM (Pin27). ???
7. Programmiertes Flash einsetzen.

Umbau des Funkteils

1. RX-VCO: Drahtschleife 11 mm einfügen.
2. TX-VCO: 2 Kondensatoren ändern bzw. hinzufügen
3. Schaltspannung für PIN-Diodenschalter abgreifen.
4. Abgleich

Ad 1.1
Der TX-VCO muss nach Umbau zwischen 451,4 MHz und 471,4 MHz (original 482,41-487,14) sicher anschwingen. Die Abstimmungspannung läßt sich an der 4-poligen Prüfbuchse auf der FT-Platine abgreifen. Ich habe vor dem Umbau die die Grenzwerte der Abstimmungspannung notiert (inzwischen aber leider verlegt).
Wie beim C5 sollten die alten Werte nach dem Umbau nicht wesentlich überschritten werden. Das Abstimm-C sollte auch keinesfalls weiter eingeschraubt werden müssen, um den den Abstimmbereich zu erreichen, eher sollte es weiter herausgedreht werden. Dadurch wird das L/C-Verhältnis im Hinblick auf die Abstimmspannung günstiger.
Ad 1.2
Der TX-VCO hat leider einen Helixkreis, von dem ich nicht weiß, wie man ihn modifizieren kann. Daher habe ich die beiden Cs C345 und C346 verändert. C346 liegt in Serie zur Abstimmdiode und wird auf 16 pf erhöht. Dadurch sinkt erstens die Resonanzbereich des Kreises, zweitens erhöht sich dadurch der Einfluss der Abstimmdiode auf die Resonanzfrequenz. C345 liegt parallel zu Abstimmdiode und C346 und wurde (glaube ich) auf 5,6 pf erhöht. Das war nötig, um einen Kompromiss zwischen Variationsbreite einerseits und Mittenlage der Resonanzfrequenz andererseits zu erzielen. Die Abstimmspannung ist ebenfalls an der Prüfbuchse abzugreifen.
Ad 2.1
Wie ich dir irgendwann schon einmal geschrieben habe, wird mit dem POWON-Signal nicht nur der Sendezug des Funkteils ein- und ausgeschaltet, sondern leider auch die komplette NF-Aufbereitung im Empfangsfall. Daher muss letztere von diesem Signal abgetrennt und dauerhaft eingeschaltet werden. Aufgrund der Leitungsführung sind dazu leider eine Litze auf der Unterseite der Steuerung und eine Leiterbahnunterbrechung sowie ein Stück Fädeldraht auf der Oberseite erforderlich.
Ad. 2.2
Das ist ein Notbehelf. Das VSDB-Signal wird eigentlich von der Datensteuerung (DST) über das BIF geschaltet. Damit das NF-Signal für die Modulation durchgeschaltet wird, muß VSDB low sein. Geschieht das nicht, wird im Sendefall nur ein Träger erzeugt. Bisher ist es mir nicht gelungen, DST richtig in den Griff zu bekommen Daher wird das VSDB-Signal einfach nach Masse gezogen. (Zwischen dem VSDB-Ausgang am BIF und seinem Eingang in der NF-Aufbereitung sitzt noch ein 1k-Widerstand)

Das wär's im wesentlichen. Man sollte womöglich noch den Hub bzw. am Empfangsteil den NF-Pegel einstellen. Vielleicht stecken aber auch noch irgendwo ein paar Überraschungen in der Steuerung.

Konzeptdokumentation PP5BOS

Das PP5BOS ist in dem Flash in einem kleinen Block am oberen Aderessende untergebracht. Die für den Einsatz im Porty in Frage kommenden Flashtypen (z. B. 29F010, 29F002TC, 29F040) besitzen einen 16Kbyte-Sektor (29F002TC) oder einen 64Kbyte-Sektor (29F040) am oberen Aderessende. Bei Bedarf kann das gesamte Inhalt neugeschrieben werden, was jedoch kritische Momente erfordert (PP5BOS einen Moment lang nur noch im SRAM)!

Betriebssystemerfordernisse:

- Einfache Aufgaben werden über den Timer0 erledigt
- Application Interface (API) über Softints (ähnlich MSDOS) mit Parameterübergabe in den Registern
- integrierter Bootloader
- Funktionen:
    - Test, ob ein Programm im RAM des Porty geladen ist
    - Test, ob Benutzerprogramm (PP5PRG) im Flash geladen ist
    - Kopiert bei Bedarf PP5BOS vom Flash ins RAM und führt es dort aus (Programmiermode)
    - PCP: Programm laden/programmieren und Update PP5BOS über externe serielle Schnittstelle
    - PCP: Aktivierung bei laufender Anwendersoftware über die serielle Schnittstelle möglich.
- Unterteilung
    - Bootloader (BL) als Bestandteil des PP5OS
    - Application Interface (API) als Bestandteil des PP5BOS
    - Signalisierung über Soft-Interrupts (Handler) durch das PP5BOS
    - Porty-Programm (PP5PRG)
    - Texte und Funktionen, Menü, Ser. IO etc.

technische Beschreibung des PP5BOS

Funktionen und Fähigkeiten des PP5BOS und des Bootloaders

Nach dem Drücken des Einschaltknopfes am Porty springt der Prozessor den Reset-Vektor (0FFFF0h) an. An ihm befindet sich ein far-jump, der auf eine Reset-Routine zeigt. In dieser Routine werden alle wichtigen Speicherbereiche und benötigte Peripherie initialisiert und eine Startmeldung zum Hörer geschickt. Danach wird der RAM an der Adresse 800h auf die Kennung „PP5PRG“ hin überprüft, ist sie vorhanden, so startet PP5BOS das Programm (Aufruf Adr. 810h). Ist keine Programmkennung vorhanden, so sucht das Programm an den 64k-Segmentanfängen im Flash nach der Kennung „PP5PRG“. Die Flash-Größe wird über eine Datenspiegelung detektiert (Daten spiegeln sich beim 128K und 256K-Modell). Findet der Bootloader eine Kennung, so überprüft PP5BOS, ob sich noch Daten des Programmes im RAM befinden (Test 8Byte 808h = FlashSeg:008h gleich). Wenn dem so ist, so führt er das Programm, das dort abgelegt ist aus. Ansonsten kopiert PP5BOS den Datenanteil in den RAM und versieht ihm mit der Kennung „PP5DAT“. Wurde kein Programm gefunden, wird der Bootloader gestartet, das dem Benutzer ein Update der Software erleichtern soll.

Kommunikationsprotokoll

Für eine geordnete Kommunikation über die externe serielle Schnittstelle ist das PP5BOS 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 PP5BOS benutzt werden. Die Kommunikation mit PP5BOS 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	Bootloader	- Speicher (SRAM) auslesen - Speicher beschreiben - Programm starten - Programm unterbrechen - Gerät neu starten, 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)

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. Die Funktionen des PP5BOS sind in 4 Funktionsgruppen unterteilt:

1. Steuerungsfunktionen PP5BOS
2. Hörer – Zugriff und Abfrage
3. Serielle I/O-Funktionen (Serielle Schnittstelle / I²C-Bus (BST) allgemein)
4. Informations- und Konvertierungsfunktionen

Nachfolgend sind die implementierten und zukünftigen Funktionen tabellarisch aufgeführt. Der I-Status (Implementierungsstatus) gibt an, ab welcher PP5BOS-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 Porty aus (INT3Bh-Aufruf)	v0.10
	01h	-	Hardware-Reset des Portys (INT3Bh-Aufruf)	
	02h	-	Softwarereset normal	v0.10
	03h	-	Softwarereset, Start des Bootloaders	v0.10
	04h	-	Startet Bootloader ohne Reset	v0.10
Interruptvector / Behandlungsroutine setzen	05h	Vec	*IN:* AL=Interruptvektor, DS:[SI]=INT-Routine, bei Behandlungsroutine ES = benutztes Datensegment	v0.10
	06h			
Reserviert	07h			
Gerät abschalten steuern	08h	00h	Ausschalten erlaubt (**default**)	v0.10
		01h	Ausschalten verboten (Taste keine Wirkung)	
Unterbrechung durch BL	09h	00h	Unterbrechungen zulassen (**default**)	v0.10
		01h	Unterbrechungen verbieten (für kritischen Code)	
Reserviert	0Ah			
Behandlungsroutine entfernen	0Bh	Vec	*IN:* AL = Vektor der Behandlungsroutine	v0.10
Warten (passives Warten)	0Eh	-	*IN:* CX: Wartezeit in (CX*10ms)	
Warten (aktive Warteschleife)	0Fh	-	*IN:* CX: Wartezeit (CX=256 → 30ms Warten)	v0.10

1xh – Hörer - Zugriff und Abfrage

Funktion	AH	AL	Beschreibung	I-Status
	10h			
Sende Text zum Hörer	11h	-	*IN:* 16Textzeichen in DS:[SI]	v0.10
Quittungston ausgeben	12h	-	Gibt einen einzelnen Ton aus	
Fehlerton ausgeben	13h	-	Gibt einen kurzen Doppelton aus	
Warte auf Hörer-Taste	14h	-	*OUT:* AL = Tastencode	v0.10
Letzte Hörertaste abfragen	15h	-	*OUT:* AL = Tastencode, AH = 0h, wenn Taste neu	v0.10
Setze Wiederholrate	16h	Val	*IN:* AL = Tastenwiederholrate in 10ms	v0.10
Setze Startverzögerung	17h	Val	*IN:* AL = Verzögerungszeit erste Wiederh. (10ms)	v0.10
Schalte Display Ein / Aus	18h	Val	*IN:* AL, Bit0 : Display Ein/Aus	v0.1d
Schreibe LEDs	19h	Val	*IN:* AL Bit0 = rote, Bit1 = gelbe, Bit2 = grüne LED	v0.1c
Schalte Mikro / Hörverstärker	1Ah	Val	*IN:* AL Bit0 = Mikro / Bit1 = Hörverstärker	v0.1c

2xh – Serielle I/O-Funktionen (Seriell und BST)

Funktion	AH	AL	Beschreibung	I-Status
Serial: Sende Zeichen	22h	Val	*IN:* Ein Zeichen, das direkt auf die Leitung geschickt wird	
Serial: Sende Paket	23h	-	*IN:* DS:[SI], CX = Paket, analog zu 22h	
Serial: Send-PCP_Short	24h	Port	*IN:* AL=Port, DL=Zeichen	v0.10
Serial: Send-PCP-Long	25h	Port	*IN:* AL=Port, DS:[SI], CX = Paket	v0.10
Sende zur Chipkarte	26h	-	*IN:* DS:[SI], CX = Paket	
Lese von Chipkarte	27h	-	*IN:* ES:[DI] = Buffer, CX = ReadLength (max)	
I²C: BST Reset	28h		Initialisiert den BST, DisplayCtrl. und KartenCtrl.	v0.12
I²C: Daten ausgeben	29h	Adr	*IN:* AL = I²C-Geräteadresse, DS:[SI], CL = Paket (Das Paket darf nur 0 bis 9? Zeichen enthalten)	v0.12
I²C: 1 Byte an Gerät schicken	2Ah	Adr	*IN:* AL = Geräteadresse, DL = Steuerbyte	v0.12
I²C: 2 Byte an Gerät schicken	2Bh	Adr	*IN:* AL = Geräteadresse, DX = Daten (DH first)	v0.12

3xh – Informations- Konvertierungsfunktionen

Funktion	AH	AL	Beschreibung	I-Status
Informationen über PP5BOS	30h	-	*OUT:* Version in AX (Hi.LOW)	v0.10
	31h	-	*OUT:* Einschaltdauer in AX, BH, BL (H, M, S)	v0.10
	33h	-	Fehlerzähler der ext. Schnittstelle	v0.10
Flashgröße bestimmen	34h	-	*OUT:* AX = Flashgröße in Kbyte	v0.10
Flashhersteller und Device-ID	35h	-	*OUT:* AH = Hersteller ID, AL = Device ID	v0.10
Interruptvector lesen (zum Sichern z.B.)	36h	Nr	*IN:* AL = Interruptvektor *OUT:* DX:AX = Adresse der Interruptbehandlungsfunktion (Seg:Ofs)	v0.10
Zahl→ASCII (DS:[DI])	3Dh	Val	*IN:* AL = Zahl, Umwandlung in 2.-Hexdarstellung *OUT:* Ascii → ES:[DI], DI = DI + 0	v0.10
	3Eh	-	*IN:* DX = Zahl, Umwandlung in 4.-Hexdarstellung *OUT:* Ascii → ES:[DI], DI = DI + 4	
	3Fh	Val	*IN:* AL = Zahl, Umwandlung in 8.-Binärdarstellung *OUT:* Ascii → ES:[DI], DI = DI + 8	

Handler

Handler sind normale FAR-Prozeduren, die durch PP5BOS aufgerufen werden Die Routinen werden analog „normalen“ Interrupt-Behandlungsroutinen mit dem „set Interruptvector“-Befehl gesetzt. Jedoch muss PP5BOS vor jedem Handler-Aufruf wissen, welches Datensegment das passende ist. Daher wird PP5BOS 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 mit Ausnahmen gesichert.:

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

INT	**Funktion**		**Beschreibung**	**I-Status**
30h . . . 39h	**well-known-Ports des PCP-Protokolles**	**00h**	KommunikationsINT (Rx) des PP5BOS	v0.10
		01h	KommunikationsINT (Rx) des DebuggingModuls	
		02h	Reserviert (Textmeldungen)	v0.10
	Ports geräteabhängiger Kanäle (PP5COM)	03h	Porty-Gerätesteuerung, Fernbedienung	
3Ah	INT-Handler für eintreffende I2C-Pakete*		Ein Paket ist vom BST empfangen worden	v0.12
3Bh	Abschalt-Handler		Das Porty wird abgeschaltet	v0.12
3Ch	Tastenbetätigung am Hörer*		*PARAM:* AL=Tastencode, AH= Betätigungszähler	v0.12
3Dh	periodischer Handler (Timer)		keine Parameter, Aufruf alle 50ms (10Hz)	v0.12
3Eh				
3Fh				
40h bis 4Fh	Reserviert, siehe Text			

* → Wenn die Anwendersoftware den INT 3Ah benutzt, so ist zu bedenken, das Tastenbetätigungen und Helligkeitsinformationen schon von PP5BOS behandelt werden und keinen INT 3Ah erzeugen. Für Tastenbetätigungen sollte die Anwendersoftware zusätzlich den INT 3Ch auf eine eigene Behandlungsroutine setzen.

Programmieranleitung

Eine Programmerstellung mit Hilfe der API-Funktionen ist ganz einfach. Zunächst muss man sich drüber im klaren ein, das im Speicher nur 30Kbyte Platz (minus Stack) zur Verfügung stehen. Bei der Programmierung in einer Hochsprache ist dieser Platz sicher recht knapp.

Speichermodell

Als Speichermodell wird Tiny (oder das kleinst mögliche) gewählt. Das Programm benötigt am Beginn einen 16Byte-langen Header, der als erste 6 Zeichen „PP5PRG“ 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 bilden die Programmkennung, die unbedingt eindeutig sein sollte (beliebig: Name, Versions- oder Prüfsummernkonstanten). Ab dem Offset 10h beginnt das PP5PRG. Das Programm wird immer so vom PP5BOS gestartet, das der Offset im Null beginnt. Die Segmentregister CS und DS zeigen beim Start auf das (gleiche) Programmsegment wenn das Programm sich im RAM befindet. Dadurch sind alle Variablen gleich verfügbar (kein „seg DATA“ etc.). 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.
Sehr große Programme (>64Kbyte), die nicht ohne FAR-Calls auskommen, müssen als „Executable“ compeliert werden und mit einem Programm „EXE-Loader“[*] für den passenden Flashspeicherbereich konvertiert werden. Das damit entstehende Image kann auch nur an die vorgegebene Stelle ins Flash geschrieben werden.

[*] Der EXE-Loader existiert noch nicht (bisher war kein Bedarf).

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 genaue Übersicht und Einzelbeschreibung der API-Funktionen wird später der Dokumentation hinzugefügt.

Benutzen eines eigenen Handlers

Ein eigener Handler ist eine einfache Funktion, die mit einem FAR-Call endet. Da das DS-Register initialisiert wurde, gibt es im Grund nichts weiter zu beachten. Da beim 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 PP5BOS zu registrieren, wird einfach die API Funktion „setintvec“ mit der passenden Handler-Nummer verwendet. PP5BOS 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 PP5BOS DS anhand dieses Bereiches wieder her. 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 Porty sollte z.B. immer einen eigenen PCP-Port benutzen. Auch mehrere um Tastatureingaben konkurrierende Programme werden hoffentlich nicht die Regel im Porty!
Ein Handler wird mit der Funktion 0Bh „entferne Handler“ wieder sauber entfernt (incl. Datensegment-Wert).

Informationen zur Porty-Hardware

In diesem Abschnitt ist in Stichpunkten zusammengetragen, was es so übers Porty 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

Alle hier zusammengetragenen Informationen wurden freundlicherweise von Guido Küppers bereitgestellt.

Speicheraufteilung

Der Adressraum des NEC-Microcontrollers beträgt 1024kByte. Das Porty besitzt 32Kbyte statischen RAM an der Adresse 0000h bis 3FFFh, welches sich jedoch an der Adresse 20000h spiegelt Hier befinden sich die Interruptvektortabelle des x86-kompatiblen NEC's, (1Kbyte) sowie beim Betrieb von PP5BOS ein weiteres KByte Daten. Am oberen Ende des RAM's befindet sich der Stack, deren Größe variabel bleibt und ca. 128Byte umfassen sollte. Die restlichen ~29Kbyte sind vollständig durch Programme nutzbar. In dem Bereich ab 40000h befinden sich verschiedene Speichereinblendungen (tnx Guido):

- **40001h**: R/W, ff – wait - 00 -> CPOFF on, AFLOOP off
- **40002h**: R/W,
- **40004h**: R/W, BST I/O Register
- **40008h**: R, Status Register
- **40010h**: W, 80 == TXKEY ON, 0 == TXKEY off
- **40020h**: W Vorgefundene Werte 1, 2, 4, 8, 10h, 20h, 40h. 2 startet Watchdog
- **40040h**: W, Handshake-Register, ZST->BST Handshake: 0/ff = low/high

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

Peripherie

Eingabeport (0)

Bit0	C0	ADC-Daten (seriell)
Bit1	EIN	Einschaltsignal vom Hörer: Solange 1, bleibt Porty eingeschaltet. Ein Tastendruck schaltet dieses Signal um. Ist das Signal 0, so schaltet das BIF nach ca. 30s das Porty ab (Watchdog). PP5BOS schaltet bei 0 sofort ab.
Bit2	C2	1, wenn +5V in Ordnung.
Bit3	C3	Pin 9 vom IC202, 80C51 (P1.7 v. Messsteuerung).
Bit4	C4	AWS, "Automat. Wahlstart", 1 bedeutet Ruhelage.
Bit5	C5	VS, "Verteilte Signalisierung", 1= Ein, bedeutet Normalbetrieb.
Bit6	C6	Timer, "Zündung ein", 1 = Zündung ist eingeschaltet.
Bit7	C7	Pin 6 von IC203, 80C51 (P1.4 v. Datensteuerung) und Pin 11 von IC202, 80C51 (P3.0 v. Messsteuerung)

Eingabeport (1)

Funktionen unbekannt. Existiert nur im FG53!

Ausgabeport (0)

Bit0	F0	CAOFF, H == "Hör-/und Sprechadern geschlossen"
Bit1	F1	CASCR, H == "Verschleierung aus"
Bit2	DSPSU	STUMM, 1= NF abgeschaltet
Bit3	F3	POWON, 1 = "Sendebetrieb"
Bit4	TONON	TONON, 1= "Hörton ein"
Bit5	F5	RUFON, 1 = "Sekundärruf durchschalten"
Bit6	F6	AFOFF, "NF-Busausgang", 0 = "aktiv"
Bit7	F7	ACCUTEST, 1 = "Accutest"

Ausgabeport (1)

Bit0	E0	SYNDA (Synthesizer Daten)
Bit1	E1	SYNCL (Synthesizer Clock)
Bit2	E2	SYNEN (Synthesizer Enable)
Bit3	E3	TXSP0 (Senderleistungspegel Bit0)
Bit4	E4	TXSP1 (Senderleistungspegel Bit1)
Bit5	E5	TXSP2 (Senderleistungspegel Bit2)
Bit6	E6	Reset für den BST-Controller (IC204, 84C40)
Bit7	E7	Pin 7 v. IC203, 80C51 (P1.5 v. Datensteuerungs-Controller)

Ausgabeport (2)

Bit0	D0	ADC DIN
Bit1	D1	ADC SCLK
Bit2	D2	ADC /CS (active low)
Bit3	D3	AUS Internes Ausschaltsignal
Bit4	D4	M5 Messpunkt
Bit5	D5	DAC /EN (active low)
Bit6	D6	DAC /CLK (active low)
Bit7	D7	DAC DIN

Speichereinblendungen vom BIF

Adresse 40001h

Lese und Schreibzugriff.
`FFh 00h` -> CPOFF on, AFLOOP off
Genaueres ungeklärt.

Adresse 40002h

Lese und Schreibzugriff.
Genaueres ungeklärt.

Adresse40004h

Lese und Schreibzugriff.
Ist BST I/O Register.

Adresse 40008h

Nur Lesezugriff.
Status von BST, MST und DST.

Bit0	Zustand der BHS-Leitung (BST->ZST Handshake). Pro Umschalten der BHS-Leitung generiert das BIF einen Interrupt.

Bit1	?, DST/MST
Bit2	?, DST/MST
Bit3	?, DST/MST
Bit4	?, DST/MST

Adresse 40010h

Nur Schreibzugriff.
An Bit7 befindet sich die Leitung TXKEY. Sie wird bei 1 ein- und bei 0 abgeschaltet.

Adresse 40020h

Nur Schreibzugriff.
(vorgefundene Werte: 1, 2, 4, 8, 10h, 20h, 40h)

Bit0	?
Bit1	1 = Watchdog eingeschaltet
Bit2	?
Bit3	DST/MST
Bit4	?
Bit5	DST/MST

Adresse 40040h

Nur Schreibzugriff.
An Bit0 befindet sich die Handshake-Leitung der BST (ZHS). Diese wird von der ZST pro übertragenen Datum umgeschaltet.

Bussteuerung (BST) und I²C-Pakete

Jegliche Datenübertragung zwischen ZST und BST beginnt mit einem Interrupt der BST. Im Falle der Übertragungsanforderung von ZST nach BST löst die ZST diesen Interrupt durch kurzes Toggeln der ZHS-Leitung (Bit0 von 40040: ->0->1) aus. Jedoch sollte vor und unmittelbar danach geprüft werden, ob der BST nicht schon ein Paket los werden möchte (BHS-Leitung LOW).

Das erste nach 40004 (IO Register) geschriebene, bzw. von dort gelesene Byte bestimmt die Art der Transaktion und die Anzahl der zu übertragenden Bytes.

Mit jedem auszugebenen oder einzulesenden Byte ist die ZHS-Leitung umzuschalten (Toggle). Ist die BHS-Leitung nach Ausgabe des letzten Bytes LOW (BST toggelt diese), so setzt der BST die Leitung wieder auf HIGH. Dies löst jedoch einen (ungewollten) Interrupt aus.

BST-Initialisierung

C0 10 07 0A	Zuweisung der Geräteadresse 10 an den BST-Controller
F4	Abfrage der I2C Adressen aller angeschlossenen Geräte. Die BST antwortet mit F7 [X] und F8 [X]. X enthält dabei die I²C-Adresse des antwortenden Geräts. F8 steht für das letzte Gerät (ab diesem Zeitpunkt können weitere Befehle gesendet werden).
D0 [X]	[X] sind Befehle für BST-Zusatzfunktionen. Folgende Befehle sind möglich: 07: I²C-Bus zum Funkteil abschalten. 17: I²C-Bus zum Funkteil anschalten. 40: Abfrage Leitung MA (Mobilstromversorgung). Antwort: F1 [00/10] an-/abgeschaltet 19: ? 08: ? 00/12: PRES1 high/low (Pin 5 Prüfschnittstelle)

BST-Meldungen

F7 [ab]	I²C-Geräteadresse (ab), Rückmeldung auf BST-Befehl F4.
F8 [ab]	I²C-Geräteadresse (ab), höchste Geräteadresse (Rückmeldungen beendet).

I²C-Datenübertragung

Von der BST zu einem I²C-Gerät:

I²C-Datenübertragung von der BST mit Adresse `[ab]` (= 10h) an das Gerät mit der I ²C-Adresse `[em]`:

`A[X] [em] [ab] [Y Y Y ... ]`	X ist die Anzahl der Daten (0 bis 9) Y ist ein Datenbyte von X Stück (0 bis 9)
`96 A6 E0`	Auslesen des EProms ab Adresse E0, Ausgabe von 7 Bytes.
`A0 4A 1C`	Adressierung des DTMF-Generators.

Von einem I²C-Gerät zur BST:

I²C-Datenübertragung von Gerät mit Adresse [ab] an BST mit Adresse [em]:

`C[X] [em] [ab] [Y Y Y ... ]`	X ist die Anzahl der Daten (0-9 Y ist ein Datenbyte von X Stück (0 bis 9)
`9[X] [ab] Y Y Y Y Y`	Antwort auf `96 A6 E0`, verwendet von EPROM Abfrage

A9 FF F5 84 50 00 00 00 F1 00 28 86: Fehlermeldung BST ?

Befehle an den Display-Controller

Der Display-Controller befindet sich im Hörer und verwaltet Display und Tastatur.
Der Displaycontroller wird mit [`29h 08h]` und [`09h 0Ah F0h]` aktiviert. Diese Aktivierung geschieht in PP5BOS oder beim Aufruf von BST-Reset. Danach wird im Sekundentakt (ca. 1,2s) die Umgebungshelligkeit (`81h - FFh`) gemeldet. Tastatureingaben werden unmittelbar gemeldet. Erhält der Controller für ca. 10 Sekunden keine Eingabe, fällt er in den Standby-Zustand zurück (das Display zeigt 16 "*") und muss neu initialisiert werden.

`08`	Display-Controller Standby? keine Meldungen mehr, reagiert nicht mehr auf Befehle
`09 [0A F0]`	Display-Controller Wakeup, Antwort: 09 50 (beim FG52 09 40), danach wird im 1,2 Sekundentakt Umgebungshelligkeit gemeldet.
`20 X X X X X X X X`	Anzeige von 8 Zeichen (X = ASCII) in der ersten Displayzeile ab Position 0.
`21 X X X X X X X X`	Anzeige von 8 Zeichen (X = ASCII) ab der aktuellen Cursor-Position.
`21 B[X]`	Blinken Xtes Zeichen in Display
`21 C[0-7]`	Xtes Symbol am Displayrand einschalten, beginnend mit 0.
`21 C[8-D]`	Graphikzeichen im Textdisplay darstellen.
`21 CE`	?
`21 CF`	Symbole ausschalten.
`21 D[0-F]`	Displayhelligkeit X = 0 dunkel, X = F hell einstellen. (PP5BOS regelt Helligkeit)
`21 E[0-D]`	Sonderzeichen
`21 F6`	Sonderzeichen
`21 FE`	Blinken beenden
`29`	? (Erscheint im I2C-Bus-Log)

Befehle an den Kartencontroller

Der Kartencontroller befindet sich beim Porty FG52 und FG53 im Hörer und verwaltet die Ansteuerung von Berechtigungskarte, LEDs, Tastaturbeleuchtung, Piepser, Mikrofon und Hörverstärker. Der Kartencontroller wird mit [29h 08h] und [09h 0Ah F0h] aktiviert. Diese Aktivierung geschieht in PP5BOS oder beim Aufruf von BST-Reset. Danach wird im Sekundentakt 0FFh gemeldet.

29	gehört zur Initialisierung
08	wie Display-Controller
09 [0A F0]	wie Display-Controller, Antwort: 09 50, danach wird im Sekundentakt ffh gemeldet
21 ff	unbekannt
23 XX	XX bits: 7 6 5 4 3 2 1 0 | | | | | | | | | | | | | | | rote LED (1=aus) | | | | | | gelbe LED (1=aus) | | | | | grüne LED (1=aus) | | | | Tastaturbeleuchtung (1=an) | | | Stromversorgung Display (1=an) | | Mikrofonverstärker (1=an) | Hörverstärker (1=an) Piepser (1=an)
24 xx	digitales Poti Hörverstärkereingang, das 2.Byte gibt die Schrittweite an (7Bit, 0..127)
26 xx	digitales Poti Hörverstärkereingang, das 2.Byte gibt die Schrittweite an (7Bit, 0..127)
50	Auslesen Karte
51	Auslesen Karte
52	Auslesen Karte

Meldungen des Kartencontrollers

FFh	?
7Eh	nach Entfernen der Karte
7Fh	nach Einlegen der Karte

Befehle an den DTMF-Geber

1 Steuerbyte enthält die Frequenzinformation (0 = stumm). Genauere Angaben siehe Datenblatt vom PCD3311.

Befehle an die Freisprecheinrichtung

1 Steuerbyte.

```
Bits:
7 6 5 4 3 2 1 0
| | | | | | | |
| | | | | | |
| | | | | |
| | | | |  Lautsprecher aus = 1
| | | |
| | |  Lautsprecher aus = 1 (?)
```

Verstärkung 0 = laut, 111 = leise

Zeichensatz des Hörers

20h - 7Dh	Standard ASCII
80h - A8h	Standard ASCII

Tastaturcode des Hörers

Der vom Bedienhörer gelieferte Tastencode ist so erst mal schlecht brauchbar:

Code	Taste	Code	Taste	Code	Taste	Code	Taste
77h	Taste gelöst, 18h	7Fh	Abgehoben, 0Eh	7Eh	Aufgelegt, 0Fh		
00h	'A'	01h	'B'	02h	'C'	03h	'D'
04h	'E'	05h	'*'	06h	'0'	07h	'#'
08h	'F'	09h	'G'	0Ah	'H'	0Bh	' I '
0Ch	'J'	0Dh	'7'	0Eh	'8'	0Fh	'9'
10h	'K'	11h	'L'	12h	'M'	13h	'N'
14h	'O'	15h	'4'	16h	'5'	17h	'6'
18h	'P'	19h	'Q'	1Ah	'R'	1Bh	'S'
1Ch	'T'	1Dh	'1'	1Eh	'2'	1Fh	'3'
20h	'U'	21h	'V'	22h	'W'	23h	'X'
24h	'Y'	25h	F-Taste	26h	C-Taste	27h	Hörer-Taste
28h	'Z'	29h	-/0 Taste	2Ah	<->-Taste	2Bh	→◆ Taste
2Ch	SOS-Taste	2Dh	lauter-Taste ◢	2Eh	leiser-Taste ◣	2Fh	unbekannt

Daher codiert PP5BOS die Tasten in Standart ASCII-Zeichen um. Die ASCII-Codes sind zu 99% identisch mit der in der Original-Software verwendeten Codes. Nicht Umcodiert werden die Hörer-Abgehoben und Hörer-Aufgelegt „Tasten“ 7Fh, 7Eh. Ansonsten werden die Ziffern- und Buchstabentasten in ihre ASCII-Pendants übersetzt (Großbuchstaben). Die Meldung Taste gelöst wird durch setzen des 7ten Bits des letzten Tastencodes übermittelt. Für die Sondertasten wird folgende Zuordnung verwendet:

Taste	Code	ASCII	Taste	Code	ASCII
F-Taste	24h	'$'	C-Taste	25h	'%'
Hörer	0Dh	LF	-/0	20h	' '
<->	1Ah	-	→◆	3Fh	'?'
SOS	21h	'!'	Lauter◢	2Bh	'+'
◣Leiser	2Dh	'-'			

Der Empfänger

- folgt noch -