LUNDS UNIVERSITET
Institutionen för informationsteknologi
EDI021: Digitala projekt

Nummerpresentatör

Henrik Torstensson, d02ht@student.lth.se
Carl Wolff, e04cw@student.lth.se

Inlämnad den 5 december 2006

Abstract

Caller ID is nowadays a standard service in Swedish household and is now the primary way of logging incoming calls. A requirement for the caller ID is that the phone company supplies the DTMF-codes needed to decode an incoming number. It may be necessary to subscribe to this service if it is not already included with the phone line subscription.

A caller ID itself is not a very complex design. It is needed to decode the specific DTMF-tones, which is modulated in different frequencies, and present the incoming digits on a display or to a computer. An additional feature may be a call log which save each incoming phone number together with a time and date stamp.

In some countries the data from the phone company also includes the subscription owner's name. This service is not implemented in Sweden but can be achieved by using a phonebook that match each number with a name. Instead of using manual input of each phonebook entry, a connected computer could automate this by using an online number bureau. It would then be possible, upon detection of an incoming number, to find the name and save this to the onboard phonebook.

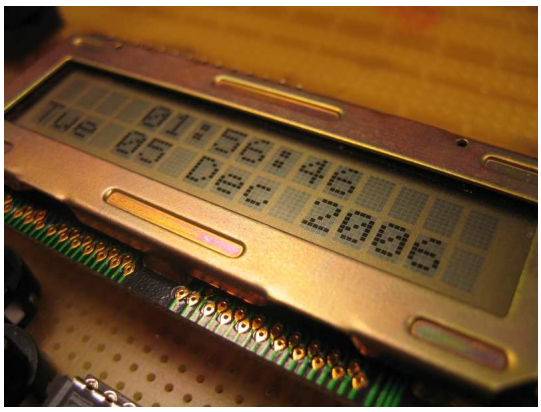
Innehåll

1 Inledning

Kursen Digitala Projekt, som ges på Lunds tekniska högskola vid institutionen för informationsteknologi, innebär genomförandet av ett projekt från kravspecifikation till konstruktion och rapport.

Vårt mål med projektet var att konstruera en nummerpresenatör som kan administreras med hjälp av en ansluten dator.

Vår kravspecifikation innefattade många delar och funktioner och vi insåg först under utvecklingen att det är mycket att hinna med på kort tid, speciellt då många timmar går åt exempelvis till felsökning. Resultatet av vår nummerpresentatör blev en fungerande prototyp för nummerpresentation där man även via dator aviseras ett inkommande samtal.



Figur 1: Nummerpresentatören i viloläge

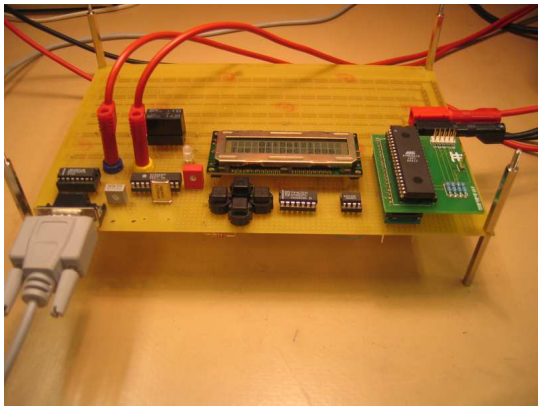
2 Specifikation

Här följer en lista med funktioner som är implementerade. Funktioner som saknas, och som finns med i den ursprungliga kravspecifikationen, kommenteras i avsnitt 5.

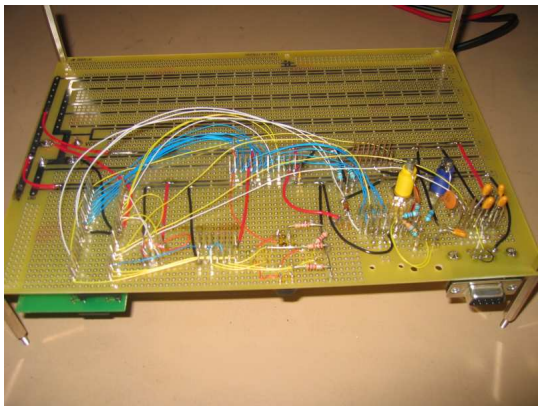
- Detektera inkommande telefonnummer och spara dessa i en samtalslista
- Avisera inkommande telefonnummer till display
- Avisera inkommande telefonnummer till dator
- Klocka med datum och tid för stämpling av samtal samt för visning i display
- Telefonbok för namnvisning vid inkommande nummer eller då bläddring sker i samtalslista
- Tryckknappar för att navigera menyn, samtalslistan, ställa datum och tid med mera
- Datorkommunikation med hjälp av ett CLI-gränssnitt där enradiga kommandon används för att administrera telefonbok och samtalslista

3 Genomförande

I detta avsnitt behandlas genomförandet av projektet; först hårdvarudelen och därefter mjukvaran. Figur 2 visar nummerpresentatören från ovansidan, och figur 3 från undersidan.



Figur 2: Nummerpresentatören



Figur 3: Undersidan av nummerpresentatören

3.1 Hårdvara

3.1.1 Huvudkomponenter

Konstruktionen är baserad på mikroprocessorn Atmel ATmega16, en relativt avancerad enchipsdator med många funktioner såsom inbyggd on-chip debugging och gränssnitt för kommunikation via USART.

Konstruktionen bygger i huvudsak på följande komponenter:

- Atmel ATmega16, AVR-mikroprocessor
- Sharp Dot-Matrix LM162XXX, LCD-display med 16x2 tecken

- Mitel MT8880C, DTMF-transceiver
- Maxim MAX232, USART för seriell PC-kommunikation med V.28/V.24
- Xicor X25128, EEPROM-minne 16k x 8 bitar via SPI-gränssnitt
- Tryckknappar för riktning – sluter vid knapptryck (NO)

För fullständig komponentlista, se appendix E. Konstruktionen i form av ett kretsschema finns i appendix 6.

3.1.2 Mikroprocessor

Nummerpresentatörens huvudkomponent, ATmega16, hanterar all inkommande och utgående data för att sedan bland annat presentera information på en LCD-display och via datorkommunikationsgränssnittet. Klockfrekvensen är maximalt 16 MHz, men har skalats ned till 1 MHz då detta är tillräckligt för att hantera de beräkningar och den datamängd som används i nummerpresentatören.

Enheter för inkommande data:

- DTMF-transceiver – i läge för mottagning av DTMF-toner.
- Datorkommunikation – kommandon från programvara
- Tryckknappar

Enheter för utgående data:

- LCD-display
- Datorkommunikation – för att tillhandahålla ett kommandogränssnitt
- DTMF-transceiver – i läge för sändning av DTMF-toner

Mikroprocessorn har 40 ben där en del är reserverade för exempelvis strömmatning, men 32 ben är allmänna för användning av in- eller utdata. Det finns i huvudsak fyra (4) portar med åtta (8) ben vardera. Då till exempel ett debuggingsystem är nödvändigt under utvecklingsstadiet är ett antal reserverade för detta. De övriga används åt konstruktionen.

Benen är i huvudsak fördelade på display (LCD), externt minne (EEPROM), DTMF-transceiver, datorkommunikation (USART), lysdioder och tryckknappar. Eftersom både LCD- och DTMF-kretsen kräver relativt många ben ligger dessa på en gemensam buss och aktiveras var för sig med kretsval (eng. *chip select*).

3.1.3 DTMF-transceiver

För nummerpresentatörens grundläggande funktion, nummervisning, används MT8880C som är en enhet för att både kunna skicka och ta emot DTMF-toner. Vid inkommande samtal, som detekteras med hjälp av Telia Soneras givna regler, ställer sig enheten beredd att detektera varje siffra och sedan spara den i ett internt register . Så fort data finns redo i registret skickas ett avbrott till mikroprocessorn, som hämtar datan.

DTMF-transceivern har, utöver nummeravkodning, också möjlighet att generera DTMF-toner. Detta kan användas för att ringa upp telefonnummer från exempelvis samtalslistan.

3.1.4 LCD-display

LCD-displayen, LM162XXX, som används för att visa inkommande nummer, meny, telefonbok och klocka, är tvåradig och kan presentera 16 tecken per rad. Den korta radlängden utgör ett problem då en del namn ej kan visas med sin fulla längd ,vilket därför kräver en mjukvarumässig åtgärd, se avsnitt 3.2.2.

3.1.5 Minne

Den mikroprocessor som används för projektet har ett starkt begränsat minne då man inte bara vill lagra programkod. För att kunna lagra många inkommande nummer samt en telefonbok med många namn krävs det att man använder ett externt minne, där det finns en uppsjö av minneskretsar att välja mellan. De EEPROM-minnen som oftast används utnyttjar parallell datainmatning. Detta skulle kräva närmare 20 ben på mikroprocessorn. För att använda färre ben krävs ett seriellt minne med exempelvis SPI-gränssnitt, och ATmega16 har stöd för detta. Det krävs då enbart fyra (4) ben men kräver implementering i form av användning av SPI-gränssnittet.

Minneskretsen har åtta (8) ben där fyra (4) används för styrning och dataöverföring. Data överförs i serie med hjälp av pinnar för Serial Output (SO) och Serial Input (SI). Inklockningen sker med hjälp av den klocka som används för SPI-gränssnittet på mikroprocessorn, Serial Clock (SCK). Utöver detta används ett ben för kretsval.

3.1.6 Datorkommunikation

Konstruktion har möjlighet att skicka ett nummer till en dator, där programvara kan utföra olika åtgärder med det, såsom att spara det i telefonboken tillsammans med ett namn. Nummerpresentatören har alla funktioner för att fungera individuellt, men dess funktion stärks då den är kopplad till en dator, då den kan styras med hjälp av ett CLI (kommandogränssnitt).

För att kommunicera med datorn används RS232 som idag är en industristandard för seriell kommunikation. Dess stora fördel är den enkla implementationen, dess nackdel är hastigheten, vilket här saknar relevans då datamängden är liten.

3.1.7 Tryckknappar

Då en knapp trycks in kan detta lätt uppfattas som flera knapptryckningar på grund av mekaniska *knappstudsar*. För att undvika detta används ofta en kondensator men kan istället hanteras i mjukvara, se avsnitt 3.2.6.

3.2 Mjukvara

Mjukvaran, skriven i programspråket C, är utvecklad för att enkelt kunna implementera nya funktioner. I varje specifik modul, för varje enhet, finns funktioner för att utföra de mest grundläggande åtgärderna, vilka används för att till exempel skriva funktioner som utför en mer övergripande procedur. Funktioner, såsom läsning från EEPROM-minnet, läser en byte ett bestämt antal gånger.

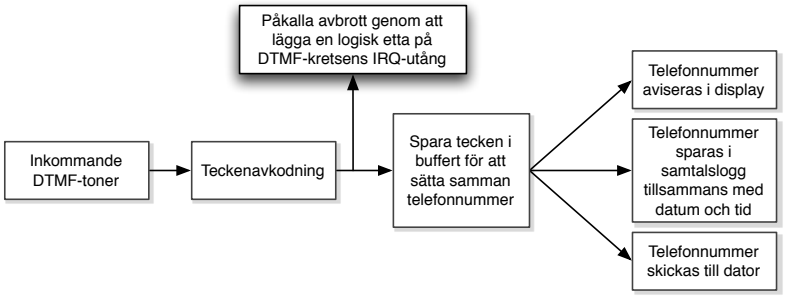
3.2.1 DTMF

Telenätet använder en speciell sekvens av spänningsförändringar inför ett kommande samtal. Detta utlöser händelser, såsom att en telefon ringer eller att en nummerpresentatör ställer sig i läge för att avkoda DTMF-toner som sänds precis innan första ringningen. DTMF-koderna representeras som siffror i dataregistret på DTMF-kretsen.

När en inkommande siffra väntar i MT8880C:s register skickas ett avbrott till mikroprocessorn, som i sin tur läser av siffran från kretsen. Det finns tre fall för inkommande samtal:

- Telefonnummer – lagras som en sifferföljd i en buffert
- Skyddat nummer – representeras som 10
- Okänt nummer – representeras som 00

De tre fallen kommer att resultera i att ett flertal händelser utförs. Telefonnummer kommer att sparas i EEPROM-minnet, skickas till dator samt aviseras i display. Detta visas i figur 4.



Figur 4: Händelseförlopp vid telefonnummermottagning

3.2.2 LCD

Displayen används för att visa inkommande samtal, datum och tid, samtalslista, meny och telefonbok. Denna, tillsammans med tryckknapparna, möjliggör menynavigation.

Teckenfönstrets storlek är två (2) rader med vardera 16 tecken. Det finns en intern buffert för längre text då man, med hjälp av funktioner i displayen kan flytta hela teckenfönstrets innehåll till höger och till vänster. Dessvärre kan varje rad ej scrollas individuellt, men detta möjliggörs med hjälp av en radscrollningsfunktion i mjukvaran. Denna används för att få nedre raden att vara fast, samtidigt som den övre visar telefonnummer och, om möjligt, tillhörande namn enligt formatet

```
Förnamn Efternamn <0701123456>
001 13:37 01 Dec
```

Då visning av samtalslista är aktiv sker förflyttning till en tidigare post med en knapptryckning uppåt och till en senare inkommen, med en knapptryckning nedåt.

Postbeskrivning	Adressintervall	Antal	Storlek	Total minnesanvänding
Samtalslista	0x0000-0x0C4D	150 st	21 byte	3 150 bytes
Telefonbok	0x0C4E-0x3FE5	281 st	47 byte	13 207 bytes

Tabell 1: Uppdelning och adresseringsintervall i externt EEPROM-minne

Bit	7	6	5	4	3	2	1	0
Innehåll	0	0	0	Skyddat	Okänt	Missat	Riktning	Data finns

Tabell 2: Flagga för en samtalspost

3.2.3 Minne

Det externa minne som finns i nummerpresentatören används för att lagra samtalslista samt telefonbok. Minnet är uppdelat i två delar, en för samtalslista och en för telefonbok. Se tabell 1.

Varje post i samtalslistan sparas tillsammans med flagga, nummer och tidpunkt, se tabell 2. I tabell 3 visas flaggan, 1 byte stor, som innehåller egenskaper såsom beskrivning om en post blivit läst, alltså visad i samtalslogg, om numret är okänt eller skyddat, samt samtalsriktning (inkommande eller utgående). Flaggan innehåller även information om huruvida posten ska anses vara raderad, för att därmed inte behöva radera hela posten,som vid radering av filer i många filsystem.

Flaggan för en samtalspost innehåller information om ett nummer är skyddat (bit 4) eller okänt (bit 3), samtalet ej tidigare har visats i display (bit 2), om det är inkommande eller utgående (bit 1) och om posten finns eller ifall den har raderats (bit 0). De första tre bitarna (bit 7, 6 och 5) är alltid 0.

Samtalsposten innehåller flaggan (1 byte), telefonnummer (16 bytes) och tidpunkt (4 bytes).

Vid skapandet av en ny post läggs den efter föregående post. Ifall en post raderas kvarstår en tom lucka, vilket indikeras i flaggan med nolla i bit 0, ”Data finns”. Detta möjliggör, en ej implementerad funktion, att utföra en defragmentering av minnet där poster flyttas nedåt i minnesadresserna, då gränsen för antal poster är uppnådd.

3.2.4 Datorkommunikation

All kommunikation med nummerpresentatören kan ske via ett CLI, ett kommandogränssnitt. Det finns ett antal kommandon som kan användas för att utföra funktioner såsom att spara,

Flagga (1 byte)	Telefonnummer (16 bytes)	Datum och tid (4 bytes)
3	30373031313233343536000000000000	45702FFC

Tabell 3: Beskrivning av en samtalpost från ett missat inkommande samtal med telefonnummer 0701123456 och där datum och tid är 1 december 2006 kl. 13:37:00 (1164978000)

Bit	7	6	5	4	3	2	1	0
Innehåll	0	0	0	0	0	0	Blockerad	Data finns

Tabell 4: Flagga för en telefonbokspost

Flagga (1 byte)	Telefonnummer (16 bytes)	Namn (30 bytes)
0	30373031313233343536000000000000	466F6F20426172 "00"x23

Tabell 5: Exempel på en ej blockerad post i telefonboken där numret är "0701123456" och namnet är "Foo Bar".

ändra eller ta bort telefonboksposter. I dess enklaste form kan dessa kommandon användas direkt i ett terminalfönster.

Genom att använda kommandogränssnittet kan programvara i datorn enkelt nyttja funktionerna i nummerpresentatören. Datorprogramvara med läsning och skrivning av data från och till serieporten kan byggas ut med funktionalitet för att hämta tillhörande namn för ett nummer, nummeravisering eller exportering av samtalslista till en databas som i sin tur kan användas i ett webbgränssnitt.

Nummerpresentatören skickar ut data via USART utan att först kontrollera ifall det finns en förbindelse. Finns en dator ansluten, med mjukvara för att hantera den data som kommer från nummerpresentatören, utnyttjas nummerpresentatören fullt ut. Om datorförbindelse saknas, eller om mjukvara ej är aktiv, fungerar nummerpresentatören helt enskilt.

Det finns ett antal fördefinierade kommandon som möjliggör administration av telefonbok och samtalslista. Om en terminalklient används och texten

```
AP 0701123456 Foo Bar
```

skickas, så kommer en telefonbokspost skapas där strängen *Foo Bar* anges som namn och *0701123456* anges som tillhörande telefonnummer.

Fördelen med att använda ett kommandogränssnitt är att det är lätt att implementera i olika programspråk då flertalet erbjuder åtkomst till serieportar i datorn. Det går lätt att skriva programkod som skriver en telefonbokspost till nummerpresentatören och med vidare implementation kan man söka nummer på Internet och spara dessa i telefonboken.

Då nummerpresentatören, utan föregående kommando, kan skriva data till serieporten måste ett program för nummeravisering konstant lyssna på indata. Vid ett inkommande nummer, som ej finns lagrat i telefonbok, skickar nummerpresentatören ut

```
IC 0701123456
```

eller då en telefonbokspost existerar

```
IC 0701123456 Foo Bar
```

3.2.5 Realtidsklocka

För att nummerpresentatören ska kunna stämpla varje samtal med tid och datum krävs en realtidsklocka, vilket innebär en klocka med exakt rätt frekvens för att kunna åstadkomma enheten en sekund. Då ingen extern klocka används har istället en klocka implementerats direkt i mikroprocessorn genom att skala ned klockfrekvensen för att fungera som realtidsklocka.

I och med att datum och tid utgör en ganska stor och besvärlig enhet, har klockan baserats på UNIX timestamp (även kallat POSIX), vilken beskriver antalet sekunder som gått sedan den 1 januari 1970 klockan 00:00:00. Varje dygn utgör alltså 86 400 sekunder. Den 1 december 2006 kl. 13:37:00 skulle alltså i UNIX timestamp representeras av heltalet 1164978000. Det krävs enbart fyra (4) bytes för att representera ett klockslag för ett visst datum, vilket är fördelaktigt då det externa minnet är begränsat.

3.2.6 Meny och användning av tryckknappar

Menyn är uppbyggd likt en modern musikspelare, där undermenyer alltid faller ut åt höger. För att bekräfta en åtgärd, eller flytta sig in i en undermeny används högerknappen och för att återgå till föregående meny, eller till slut viloläget, används vänsterknappen. Knappar i riktning upp och ned används för att byta mellan olika alternativ i menyn. De menyfunktioner som kräver en bekräftelse representerar detta med ja/nej-alternativ på högra sidan. För att bekräfta en inmatning, till exempel datum och tid, används högerknappen.

Vid knapptryckning skapas ett avbrott som anger att undersökning av nedtryckt knapp ska göras. För att undvika att ett enda knapptryck tolkas som flera används en timer som nollställs i samband med avbrott och sedan räknar rimlig tid för en knapptryckning.

Knapptryckningar som varar en längre tid kommer att registreras som en enda nedtryckning och därmed krävs ytterligare knapptryckningar för att navigera sig genom en lista.

4 Resultat

Hela hårdvarukonstruktionen bygger på att samtliga delar, ursprungligen specificerade i kravspecifikationen, skulle implementerats. Konstruktionen innehåller bland annat möjligheter för sändning av DTMF-toner och för att avisera inkommande samtal via lysdioder.

Mjukvaruimplementationen har flertalet av de funktioner vi ville att nummerpresentatören skulle ha, men funktioner för utgående samtal, minnesoptimering och ett flertal kommunikationsmöjligheter är ej klara, såsom de kommando som krävs för att exempelvis överföra eller radera samtalslista. Dessutom var det tänkt att samtliga andra kommandon, annars utförbara genom meny, skulle kunna hanteras.

Enheten är en fullfjädrad nummerpresentatör, likt de som finns i många hem, men har en välarbetad grund för att utrustas med ytterligare funktioner. Vid inkommande samtal, presenterar den som tänkt telefonnummer, vilka stämplas med datum och tid för att sparas i en samtalslista.

Nummerpresentatörens meny möjliggör att samtalslista kan raderas och klocka kan ställas, men med en telefonbok som skulle kunna administreras i menyn, skulle även uppringning eller blockering av samtal kunna ske.

Tanken var också, i mån av tid, att en programvara skulle utvecklas för att hantera de olika kommandon som finns tillgängliga samt att denna också skulle avisera inkommande telefonsamtal. Det inkommande telefonnumret skulle dessutom, ifall ett namn ej var inkluderat, försöka matchas mot ett namn med hjälp av en internetbaserad nummerbyrå. Möjligheten att spara telefonboksposter finns redan och det enda som skulle krävas är en PC-implementation.

5 Möjligheter för vidareutveckling

Grunden för nummerpresentatören har konstruerats och skrivits med avsikt att kunna vidareutvecklas. Konstruktionen innehåller delar som ej förverkligats i mjukvara men vars hårdvarukonstruktion är klar för omedelbar implementering, som till exempel reläet kopplat till telenätet.

5.1 Detektering av utgående samtal

Då konstruktionen möjliggör inkoppling av en telefon ansluten via nummerpresentatören skulle en detektering av utgående samtal kunna ske. En bit i samtalspostsflaggan möjliggör lagring av både utgående och inkommande samtal, vilket skulle kunna användas för att detektera, ett på telefonen, slaget nummer.

Detta skulle kräva att regler sätts upp för hur lång tid det bör dröja innan numret detekteras som fullständigt då annars exempelvis koder för navigering i en telefonväxel eller personliga koder skulle lagras.

Man kan även tänka sig att hårdvaran kompletteras så att telefonlinjens olika polaritetsväxlingar detekteras.

5.2 Uppringning av telefonnummer

Kretsen för DTMF, MT8880C, är en transceiver vilket innebär att den har möjlighet både sända och ta emot DTMF-toner. Då navigation genom samtalslistan görs, skulle en post kunna

förses med möjligheter för uppringning. Uppringt nummer skulle automatiskt kunna sparas i samtalslistan och flaggas som utgående samtal.

5.3 Linjeavkänning

Linjeavkänning skulle kunna förverkligas om ytterligare ett par komponenter för tillkom. Detta skulle kunna avisera en upptagen linje genom till exempel en blinkande lysdiod eller som en symbol i displayen.

5.4 Telefonboksuppdatering

För att automatisera telefonboken skulle ett program kunna användas, vars uppgift vid inkommande nummer är att hämta namn för givet nummer ifrån en nummerbyrå på Internet. Om någon ringer och telefonnumret ej är sparat i telefonboken skulle programmet kunna söka upp telefonnumret och automatiskt spara det i telefonboken. Nödvändiga kommandon finns redan i nummerpresentatören och det enda som krävs är någon form av avlyssning av serieporten samt en funktion som kan extrahera namnet från en webbsida.

Om en adressbokspost skickas via CLI under tiden motsvarande telefonnummer visas på displayen, övergår presentatören till att visa både namn och nummer.

5.5 Expandering av kommandogränssnitt

Nummerpresentatören skulle kunna användas för att avisera inkommande samtal från något IP-telefoniprogram eller ett oläst e-postmeddelande. Detta skulle enkelt kunna implementeras genom att man använder ett kommando för att bara visa en indikation och ett kommando för att indikera en pågående händelse, såsom ett inkommande samtal exempelvis i Skype. Ett kommando som till exempel

```
PCN Incoming call from Foo
```

från datorn skulle kunna få nummerpresentatören att avisera texten i displayen samtidigt som en diod blinkar. Implementationen för detta kräver inga nya extrafunktioner förutom just möjligheten att hantera det inkommande kommandot PCN.

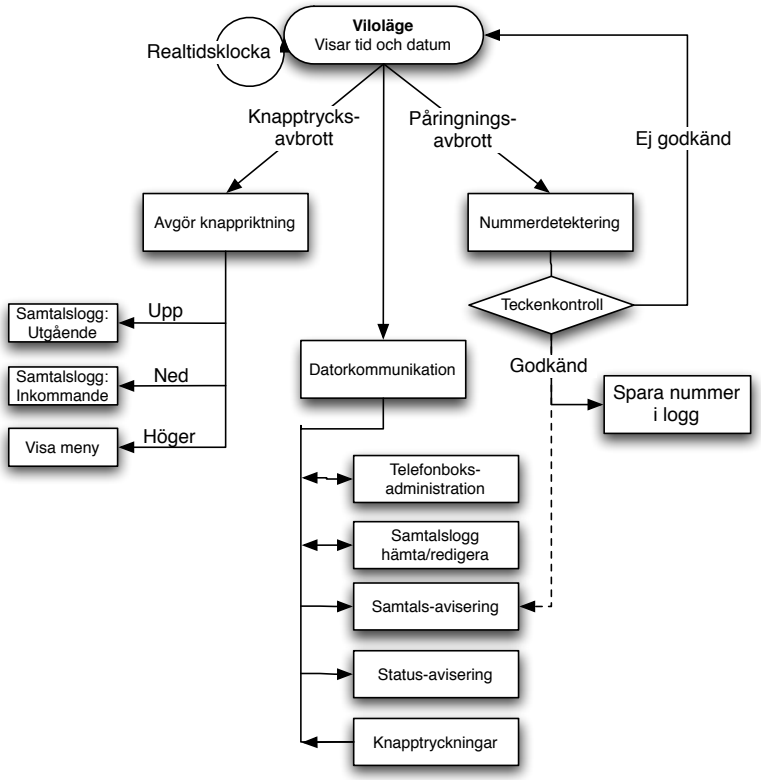
En fullständig samtalslista skulle kunna hämtas till ansluten dator för att där kunna spara en logg över inkommande samtal. Detta skulle ge möjlighet att spara en samtalslista längre bak i tiden och inte längre vara begränsad till 150 poster i samtalslistan.

5.6 Samtalslista

Samtalslistan skulle kunna effektiviseras genom att ej tidigare visade nummer, som avgörs med hjälp av flaggan i samtalsposten, visas först. Därefter kommer alla tidigare visade nummer efter en avgränsare. Detta kräver en implementering där samtalslistan först bestämmer vilka som ej visats och visar dessa. Då stegning sker genom ej tidigare visade poster förändras deras bit i flaggan och de kommer vid nästa listbläddring att grupperas med tidigare visade.

A Flödesdiagram

I figur 5 visas en enkel flödesskiss.



Figur 5: Flödesdiagram

B Serieportsinställningar

- Överföringshastighet: 4800 baud
- Databitar: 8
- Paritet: Ingen
- Stoppbit: 1

C Strängar sparade i EEPROM

Alla strängar sparade i EEPROM listas i tabell 6 och tabell 7. En pekare till en kopia av en sträng erhålles exempelvis med kommandot `str(STR_TOO_FEW_ARGS)`.

define-konstant	Sträng
STR_S_S_N	%s %s %n
STR_AP	AP
STR_TOO_FEW_ARGS	TOO FEW ARGS
STR_R_N	\r\n
STR_PERSON_ADDED	PERSON ADDED
STR_PERSON_OVERWRITTEN	PERSON OVERWRITTEN
STR_DP	DP
STR_DELETED	DELETED
STR_NOT_FOUND	NOT FOUND
STR_QP	QP
STR_UNKN_S_R_N	UNKNOWN: %s\r\n
STR_PARSE_ERROR	PARSE ERROR
STR_MENU_CHOICE1	%d %-13s\176
STR_MENU_CHOICE2	%d %-13s
STR_ARROW_EXIT	\177 Exit
STR_NO	NO
STR_YES	YES
STR_WAIT	WAIT...
STR_DONE	DONE!
STR_DELETE_ALL	DELETE ALL

Tabell 6: Strängar sparade i EEPROM, del 1

define-konstant	Sträng
STR_INCOMING	INCOMING?
STR_UNKNOWN	<Unknown>
STR_HIDDEN	<Hidden>
STR_TIME	%02d:%02d:%02d
STR_LT_S_GT	<%s>
STR_TIME_DATE	%03d %02d:%02d %02d %s
STR_NO_INCOM	No incom-
STR_ING_CALLS	ing calls
STR_DATE	%s %02d %s %04d
STR_SET_DATE	-Set date-
STR_SET_TIME	-Set time-
STR_ARROW_EXIT2	\177Exit
STR_SETUP_DATE	%04d-%02d-%02d
STR_ARROW_TIME	\176Time
STR_ARROW_SET	\176Set
STR_PHONEBOOK	PHONEBOOK
STR_BLOCKED_NBRS	BLOCKED NBRS
STR_DEL_INCOMING	DEL INCOMING
STR_DEL_OUTGOING	DEL OUTGOING
STR_SET_DATE_TIME	SET DATE/TIME

Tabell 7: Strängar sparade i EEPROM, del 2

C.1 Rådata i EEPROM

Här följer innehållet i processorns interna EEPROM i Intel hex-format. Minnet innehåller alla strängar som skrivs ut av programmet.

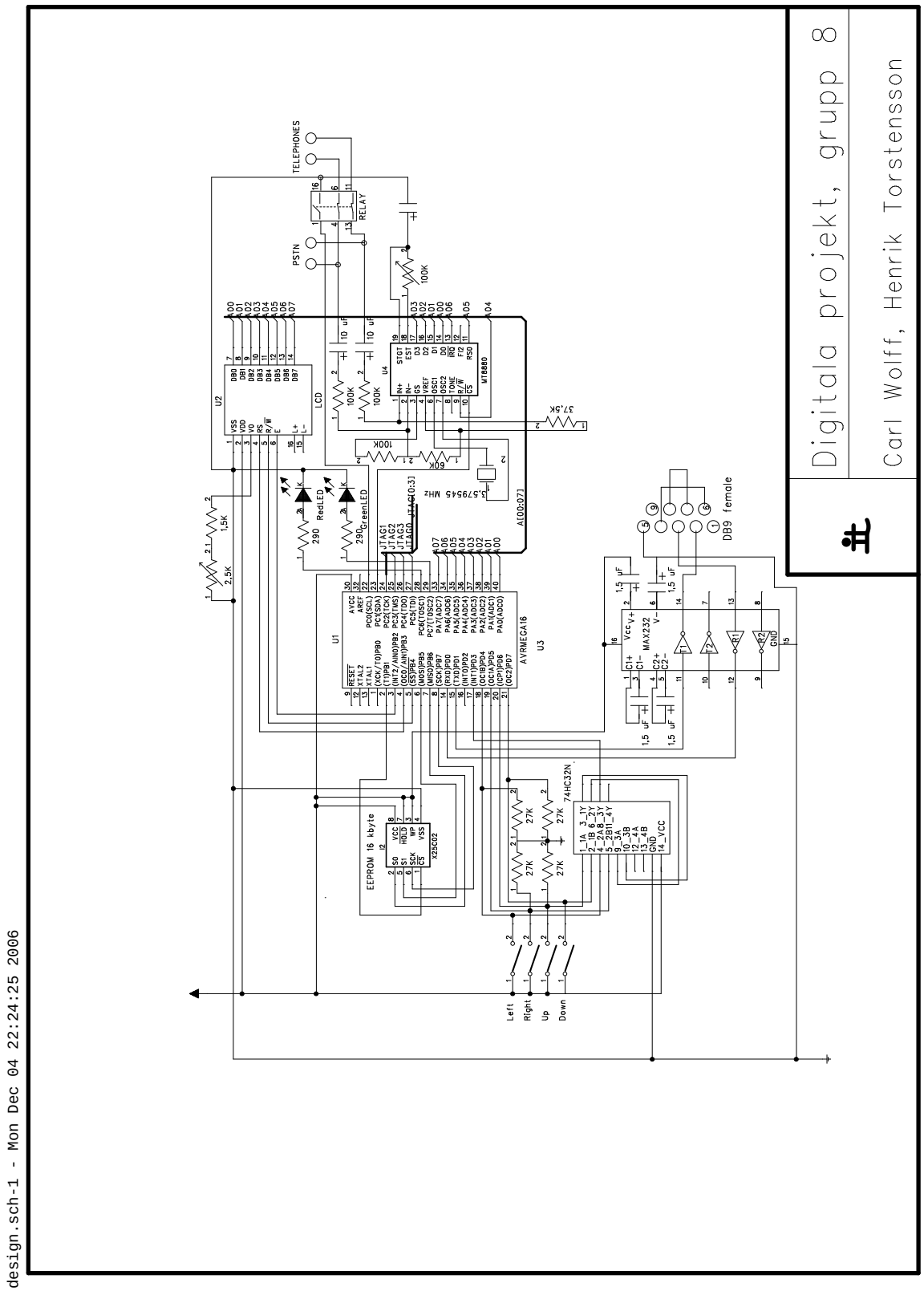
```
:10000000257320257320256E00415000544F4F204A
:1000100046455720415247530000D0A005045504570
:1000200052534F4E20414444454400504552534F93
:100030004E204F56455257524954544 54E00445055
:10004000044454C45544544004E4F5420464F55BE
:100050004E4400515000554E4B4E4F574E3A2025BE
:10006000730D0A00504150415253452045525 24FA2
:1000700052002564202 52D3133737E002564202510
:10008000642025 2D313373007F2045786974004E3C
:100090004F004E4F005945530057414954 2E2E2EC4
:1000A00000444F4E45210044454C45544520414CA9
:1000B0004C00494E434F4D494E473F003C556E6BF7
:1000C0006E6F776E3E003C48696464656E3E002545
:1000D0003032643A253032643A2530326400203CB4
:1000E00025733E00253033642025303264 3A2530B4
:1000F000326420253032642025730 4E6F20696EF3
:10010000636F6D2D00696E672063616C6C730025F1
:10011000732025303264202573202530346400 2D6F
:10012000536574206461746 52D002D5365742074CB
:10013000696D652D00207F45786974002530342570
:10014000303464 2D253032642D2530326400207E19
:10015000546 96D650020202020202020207E53657486
:1001600000454550504 4F4E45424F4F4B00424C82
:100170004F434B4544204E4252530044454C204986
:100180004E434F4D494E470044454C204F55544730
:100190004F494E4700534554204441544 52F54493C
:1001A0004D4500FFFFFFFFFFFFFFFFFFFFFFFFCA
:1001B000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFF4F
:1001C000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFF3F
:1001D000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFF2F
:1001E000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFF1F
:1001F000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFF0F
:00000001FF
```

D Kretsschema

Kretsschema visas i figur 6.

E Komponentlista

I tabell 8 visas vilka komponenter som används.



Figur 6: Kretsschema

Komponent	Antal	Beskrivning / användning
Atmel ATmega16	1	AVR 8-bits mikroprocessor
Sharp Dot-Matrix LM162XXX	1	LCD-display med 16x2 tecken
Mitel MT8880C	1	DTMF-transceiver
Maxim MAX232	1	V.28/V.24 seriekommunikation
Xicor X25128	1	EEPROM-minne, 16 KB, SPI-kommunikation
74HC32N	1	Quad (4 st) 2-input OR-grind
Tryckknappar	4	Sluter krets vid knapptryck, används för menyval
LED, röd/grön	1	Tvåfärgad lysdiod
Relä	1	För att koppla bort efterkommande telefoner
Resistor 27 kΩ	4	Ger en definierad nolla vid tryckknappar
Resistor 1,5 kΩ	1	Används för kontrastinställning av LCD
Resistor 100 kΩ	3	Används för DTMF-kretsen
Resistor 60 kΩ	1	Används för DTMF-kretsen
Resistor 37,5 kΩ	1	Används för DTMF-kretsen
Vridpotentiometer 2,5 kΩ	1	Används för kontrastinställning av LCD
Vridpotentiometer 100 kΩ	1	För finjustering av DTMF-kretsens känslighet
Kondensator 1,5μF	4	Används för MAX232-kretsen
Kondensator 10μF	2	Används för DTMF-kretsen
Kondensator		Används för MAX232-kretsen
Kontaktdon DB9 hona	1	Används för serieportskommunikation
Kontaktdon banankontakt	6	Strömförsörjning, två telefonlinjer (in och vidare)
Kristall 3,579545 MHz	1	Används för DTMF-kretsen

Tabell 8: Komponentlista

F Ursprunglig kravspecifikation

F.1 Hårdvara

Nummerpresentatören skall innehålla följande enheter:

- Display med storleken 2×20
- Fyra tryckknappar, en i vardera riktningen upp, ner, höger, vänster
- Ett RJ11-uttag för inkoppling till telefonnätet
- Eventuellt ett extra uttag för inkoppling av telefon
- En DB9-hona för seriell kommunikation med RS232
- Uttag för extern strömförsörjning alternativt plats för batteri
- Flerfärgsdiod för statusindikation

F.2 Funktionskrav

F.2.1 Nummerpresentatören

- Enheten skall detektera om en telefonlur lyfts och indikera detta i displayen.
- Enheten skall reagera på både inkommande och utgående samtal och spara numret i en samtalslista. Samtidigt visas numren i displayen.
- Samtalslistan skall skilja på inkommande och utgående samtal.
- En flerfärgsdiod skall indikera missade samtal, status på det inkommande numret (blockerat) etcetera.
- Tryckknapparna skall navigera i samtalslistan och även i en meny där man kan radera poster, visa telefonbok, blockera nummer, samt ställa in datum och tid med mera.
- Samtalslistan skall sorteras i kronologisk ordning och visa om det aktuella numret var ett inkommande eller utgående samtal. Om numret finns med i telefonboken skall även namnet visas. Först i listan kommer nummer som tillkommit sedan föregående knapptryckning, därefter visas en avskiljningsrad innan resten av samtalen listas.
- Från telefonboken skall man kunna ringa upp och blockera respektive nummer.
- Då inget förändrats i displayen på ett tag skall enheten återgå till ett grundtillstånd där datum och tid visas.

F.2.2 Kommunikation via seriellt gränssnitt

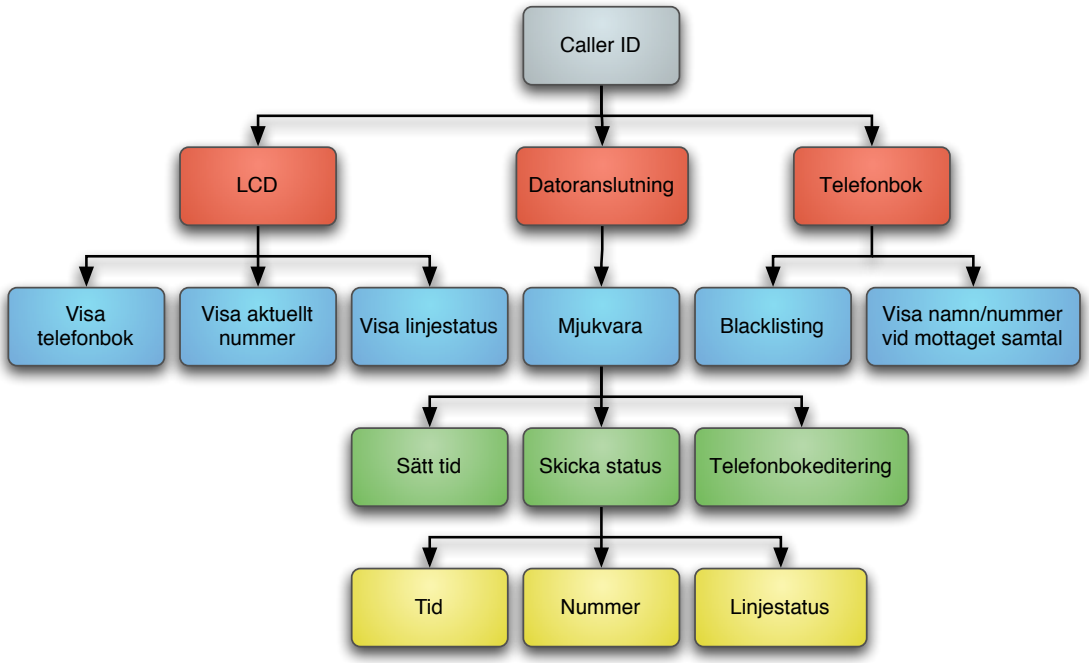
- Kommunikation med en dator skall möjliggöras via ett RS232-gränssnitt, 9600 baud, 8 databitar, ingen paritetsbit, en stoppbit.
- Gränssnittet skall vara ett CLI. Ett kommando ges på en rad enligt en bestämd syntax, och svar erhålles på en eller flera rader. Statusmeddelanden från nummerpresentatören kan komma utan föregående förfrågan.

- Allt som visas i displayen, förutom menyval och liknande, skall även presenteras på det seriella gränssnittet.
- Allt som användaren kan påverka genom fysiska knappar på nummerpresentatören skall även kunna påverkas via det seriella gränssnittet.
- Nya poster i adressboken skall kunna matas in via gränssnittet.
- Om en ny adressbokspost läggs till under tiden displayen visar den nya postens telefonnummer, så skall displayen även visa den nya postens namn.

Gränssnittet gör det möjligt att skriva datorprogram som interagerar med nummerpresentatören. Detta kan vara användbart för nätverkskommunikation. Man kan även kommunicera med den genom ett vanligt terminalprogram, exempelvis `minicom` eller `Hyperterminalen`.

F.3 Funktioner som implementeras i mån av tid

- Samtliga Sveriges namnsdagar ska sparas i minnet. När man ringer upp en person som finns i adressboken, eller då en person i adressboken ringer, skall displayen indikera att personen har namnsdag om den har det.
- Skriva ett datorprogram som slår upp mottagna nummer på `hitta.se` och returnerar eventuellt funnet namn till nummerpresentatören.



Figur 7: Skiss över nummerpresentatörens funktioner

G Headerfiler

G.1 global.h

```
#ifndef GLOBAL_H
#define GLOBAL_H

#include "time.h"
#include <stdlib.h>

#define F_CPU 1000000UL

/* Memory addresses in EEPROM */
#define START_PERSON 3151
#define END_MEM 16358

/* For the _flags_ in call_t struct */
#define CALL_NOT_EMPTY 1
#define CALL_INCOMING 2
#define CALL_MISSED 4
#define CALL_UNKNOWN 8
#define CALL_HIDDEN 16

/* For the _flags_ in person_t struct */
#define PERSON_NOT_EMPTY 1
#define PERSON_BLOCKED 2

#define MAX_CALLS 150
#define MAX_PERSONS 281

/* Strings in internal EEPROM */
#define STR_S_S_N (void*)0
#define STR_AP (void*)9
#define STR_TOO_FEW_ARGS (void*)12
#define STR_R_N (void*)25
#define STR_PERSON_ADDED (void*)30
#define STR_PERSON_OVERWRITTEN (void*)43
#define STR_DP (void*)62
#define STR_DELETED (void*)65
#define STR_NOT_FOUND (void*)73
#define STR_QP (void*)83
#define STR_UNKN_S_R_N (void*)86
#define STR_PARSE_ERROR (void*)102
#define STR_MENU_CHOICE1 (void*)114
#define STR_MENU_CHOICE2 (void*)127
#define STR_ARROW_EXIT (void*)136
#define STR_NO (void*)146
#define STR_YES (void*)149
#define STR_WAIT (void*)153
#define STR_DONE (void*)161
#define STR_DELETE_ALL (void*)167
#define STR_INCOMING (void*)178
#define STR_UNKNOWN (void*)188
```

```
#define STR_HIDDEN (void*)198
#define STR_TIME (void*)207
#define STR_LT_S_GT (void*)222
#define STR_TIME_DATE (void*)228
#define STR_NO_INCOM (void*)251
#define STR_ING_CALLS (void*)261
#define STR_DATE (void*)271
#define STR_SET_DATE (void*)287
#define STR_SET_TIME (void*)298
#define STR_ARROW_EXIT2 (void*)309
#define STR_SETUP_DATE (void*)319
#define STR_ARROW_TIME (void*)334
#define STR_ARROW_SET (void*)344
#define STR_PHONEBOOK (void*)356
#define STR_BLOCKED_NBRS (void*)366
#define STR_DEL_INCOMING (void*)379
#define STR_DEL_OUTGOING (void*)392
#define STR_SET_DATE_TIME (void*)405


extern char countdown_incoming;
extern char countdown_menu;
extern char show_time;
extern char setup_state;

/* Data structure for saving calls */
struct call_t
{
    char flags;
    char number[16];
    time_t time;
};

/* Data structure for saving persons */
struct person_t
{
    char flags;
    char number[16];
    char name[30];
};


/*
 * Display the menu.
 */
void display_menu();

/*
 * Step one item upwards in the menu.
 */
void menu_up();

/*
```

```
 * Step one item downwards in the menu.
 */
void menu_down();

/*
 * Handle the users menu action.
 */
void menu_action();

/*
 * Show the menu.
 */
void enter_menu();

/*
 * Exit the menu and display the clock.
 */
void exit_menu();

/*
 * Print a "YES"/"NO" choice at right in LCD, and indicate
 * selected choice ("yes_no").
 */
void print_yes_no();

/*
 * Change yes to no and vice versa in "YES"/"NO" choice.
 */
void change_yes_no();

/*
 * Delete all calls from EEPROM.
 */
void delete_incoming();

/*
 * Go back to menu. Restore button actions and print the menu.
 */
void back_to_menu();

/*
 * Do what user wants (delete calls / back to menu).
 */
void delete_incoming_action();

/*
 * Show incoming call on LCD and save it to EEPROM.
 */
void incoming_call(const char *infocode, const char *number);

/*
 * Loop through EEPROM, count items and initialize variables.
 * Should be called on start-up before accessing calls or persons.
```

```
 */
void memstat();

/*
 * Save the call "*call" to EEPROM.
 */
void save_call(const struct call_t *call);

/*
 * Perform one step of the scrolling.
 */
void do_scroll();

/*
 * Initialize the scroller.
 */
void start_scroll(const char *line1, const char *line2);

/*
 * Display the call chosen by "call_choice".
 */
void display_call();

/*
 * Step one item upwards in the call list
 */
void call_up();

/*
 * Step one item downwards in the call list
 */
void call_down();

/*
 * Show incoming calls.
 */
void show_incoming();

/*
 * Parse the command and arguments stored in USART buffer
 * (usart_ibuf) and take proper action.
 */
void parse_command();

/*
 * Find person with number "*number" in EEPROM.
 * Save found information to "*flags" and "*name".
 * Return the persons address in EEPROM.
 */
int find_number(const char *number, char *flags, char *name);

/*
 * Get pointer to the string represented by "*addr".
```

```
 */
char *str(const char *addr);

#endif
```

G.2 buttons.h

```
#ifndef BUTTONS_H
#define BUTTONS_H

#include <avr/io.h>

// Define hardware pins for the buttons
#define KEY_BIT_UP (1<<PD6)
#define KEY_BIT_DOWN (1<<PD7)
#define KEY_BIT_LEFT (1<<PD4)
#define KEY_BIT_RIGHT (1<<PD5)

// Define values for vector of pressed buttons
#define KEY_NONE 0
#define KEY_UP 1
#define KEY_DOWN 2
#define KEY_LEFT 4
#define KEY_RIGHT 8

extern char btn_pressed;
extern char count;

/*
 * Set the function to be called when button UP is pressed.
 */
void set_up_function(void (*func)());

/*
 * Set the function to be called when button DOWN is pressed.
 */
void set_down_function(void (*func)());

/*
 * Set the function to be called when button LEFT is pressed.
 */
void set_left_function(void (*func)());

/*
 * Set the function to be called when button RIGHT is pressed.
 */
void set_right_function(void (*func)());

/*
 * Initialization routine for buttons.
 */
void init_buttons();
```

```
/*
 * Get which keys are pressed
 */
char get_pressed_key();

/*
 * Check if the particular key "key" is pressed
 */
char key_pressed(char key);

#endif
```

G.3 clock.h

```
#ifndef CLOCK_H
#define CLOCK_H

#include "time.h"

extern time_t now;
extern struct tm *tm;

/*
 * Display the clock in the LCD.
 */
void display_clock();

/*
 * Refresh the date/time setup on LCD.
 */
void setup_refresh();

/*
 * Setup date and time.
 * "exitable" controls whether the dialog can be quitted.
 */
void setup_time(char exitable);

/*
 * Copy a tm struct to another.
 */
void clone_tm(struct tm *src, struct tm *dst);

/*
 * Enable(1)/disable(0) displaying of clock.
 */
void set_show_time(char state);

/*
 * Initialization routing for the clock.
 */
void init_clock();
```

```
#endif
```

G.4 diodes.h

```
#ifndef DIODES_H
#define DIODES_H

// Define hardware pins for the diodes
#define RED (1<<PC6)
#define GREEN (1<<PC7)


/*
 * Initialization routine for diodes
 */
void init_diodes();

#endif
```

G.5 dtmf.h

```
#ifndef DTMF_H
#define DTMF_H

#include <avr/io.h>

/* Chip pins */
#define DTMF_RW (1<<PA4)
#define DTMF_RS0 (1<<PA5)
#define DTMF_CS (1<<PC1)

/* Internal register functions */
#define W_TR 0
#define R_RR DTMF_RW
#define W_CR DTMF_RS0
#define R_SR (DTMF_RS0|DTMF_RW)

/* Control Register A */
#define RSEL (1<<3)
#define IRQ (1<<2)
#define CP_DTMF (1<<1)
#define TOUT (1<<0)

/* Control Register B */
#define CR (1<<3)
#define SD (1<<2)
#define TEST (1<<1)
#define BURST (1<<0)

/* Status Register */
#define S_DS (1<<3)
#define S_RX (1<<2)
#define S_TX (1<<1)
```

```
#define S_IRQ (1<<0)

#define DATAMASK 240 // Bits on PORTA not b0-b3
#define DTMFMASK 192 // Bits on PORTA not DTMF

/*
 * Read the status of DTMF chip.
 */
char dtmf_read_status();

/*
 * Read data from DTMF chip.
 */
int dtmf_read_data();

/*
 * Send data to DTMF chip.
 */
int dtmf_send_data(char number);

/*
 * Initialization routine for DTMF chip.
 */
void init_dtmf();

/*
 * Select DTMF chip on the bus (enable).
 */
void dtmf_select();

/*
 * Deselect DTMF chip on the bus (disable).
 */
void dtmf_deselect();

#endif
```

G.6 eeprom.h

```
#ifndef EEPROM_H
#define EEPROM_H

#include <inttypes.h>

#define EEPROM_CS (1<<PB1)

/* Status reg */
#define EEPROM_S_WPEN 0x07 // Write protect enable
#define EEPROM_S_BP1 0x03  // if protection is used
#define EEPROM_S_BP0 0x02  // if protection is used
#define EEPROM_S_WEL 0x01  // Status of write enable latch
#define EEPROM_S_WIP 0x00  // Write In Process
```

```
/* Control reg */
#define EEPROM_C_WREN 0x06  // Set the write enable latch
#define EEPROM_C_WRDI 0x04  // Reset the write enable latch
#define EEPROM_C_RDSR 0x05  // Read status reg
#define EEPROM_C_WRSR 0x01  // Write status reg
#define EEPROM_C_READ 0x03  // Read data from memory
#define EEPROM_C_WRITE 0x02 // Write data to memory


/*
 * Initialization routine for external EEPROM.
 */
void init_eeprom(void);

/*
 * Initialize the ATmega16 as master.
 */
void spi_init_as_master(void);

/*
 * Wait until SPI-transmission is complete.
 */
void spi_wait(void);

/*
 * Transmit data on SPI bus.
 */
void spi_transmit(char cData);

/*
 * Receive data on SPI bus.
 */
char spi_receive(void);

/*
 * Enable EEPROM chip.
 */
void eeprom_select(void);

/*
 * Disable EEPROM chip.
 */
void eeprom_deselect(void);

/*
 * Prepare the EEPROM for writing instructions.
 */
void eeprom_write_enable(void);

/*
 * Write data to data register.
 */
void eeprom_write_dreg(uint16_t addr, uint8_t data);
```

```
/*
 * Write data to status register.
 */
void eeprom_write_sreg(char opcode);

/*
 * Read data from data register.
 */
char eeprom_read_dreg(uint16_t addr);

/*
 * Read data from status register.
 */
char eeprom_read_sreg(void);

/*
 * Read "size" bytes from memory (address "*data")
 * and write to EEPROM on address "addr".
 */
void eeprom_from_mem(const void *data, int size, uint16_t addr);

/*
 * Read "size" bytes from EEPROM (address "addr")
 * and write to memory on address "*data"
 */
void eeprom_to_mem(uint16_t addr, int size, void *data);

#endif
```

G.7 lcd.h

```
#ifndef LCD_H
#define LCD_H

/* Define hardware pins for the LCD circuit */
#define LCD_E (1<<PB2)
#define LCD_RS (1<<PB3)
#define LCD_RW (1<<PB4)

/* Instructions */
#define LCD_CLEAR 1
#define LCD_SHIFT_LEFT 0b11000
#define LCD_SHIFT_RIGHT 0b11100
#define LCD_CURSOR_OFF 12
#define LCD_CURSOR_ON 14
#define LCD_CURSOR_BLINK 15

/*
 * Writes a character to LCD at the current address.
 */
void lcd_putchar(char data);
```

```
/*
 * Get the busy flag and address.
 */
char lcd_getaddr(void);

/*
 * Set the display's address.
 */
void lcd_setaddr(char address);

/*
 * Wait until the display is ready.
 */
void lcd_wait(void);

/*
 * Tell the display to run an instruction.
 */
void lcd_command(char command);

/*
 * Initialization routine for LCD.
 */
void init_lcd(void);

/*
 * Write string to display
 */
void lcd_write(const char* dstring);

#endif
```

G.8 time.h

```
/*
 * Copyright Droids Corporation, Microb Technology, Eirbot (2005)
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * Revision : $Id: time.h,v 1.2 2005/02/15 13:01:14 zer0 Exp $
 *
```

```
 */

#ifndef TIME_H
#define TIME_H

#ifndef __TIME_UNSIGNED
#define __TIME_UNSIGNED 1
#endif

#if __TIME_UNSIGNED
struct tm
{
  unsigned char tm_sec;         /* Seconds.     [0-60]      */
  unsigned char tm_min;         /* Minutes.     [0-59]      */
  unsigned char tm_hour;        /* Hours.       [0-23]      */
  unsigned char tm_mday;        /* Day.         [1-31]      */
  unsigned char tm_mon;         /* Month.       [0-11]      */
  int tm_year;                  /* Year since 1900          */
  unsigned char tm_wday;        /* Day of week. [0-6]       */
  int tm_yday;                  /* Days in year.[0-365]     */
  unsigned char tm_isdst;       /* Daylight saving time     */
  unsigned char tm_hundredth;   /* not standard 1/100th sec */
};
#else
struct tm
{
  int tm_sec;                   /* Seconds.     [0-60]  */
  int tm_min;                   /* Minutes.     [0-59]  */
  int tm_hour;                  /* Hours.       [0-23]  */
  int tm_mday;                  /* Day.         [1-31]  */
  int tm_mon;                   /* Month.       [0-11]  */
  int tm_year;                  /* Year since 1900      */
  int tm_wday;                  /* Day of week. [0-6]   */
  int tm_yday;                  /* Days in year.[0-365] */
  int tm_isdst;                 /* Daylight saving time */
  char *tm_zone;                /* Abbreviated timezone */
};
#endif

extern char monthDays[12];

extern char *__day[7];
extern char *__month[12];

typedef unsigned long time_t;

struct tm *gmtime(time_t *timep);
struct tm *localtime(time_t *timep);
time_t mktime(struct tm *timeptr);

#endif /* TIME_H */
```

G.9 usart.h

```
#ifndef USART_H
#define USART_H

#include <stdint.h>

extern char usart_ibuf[64];

/*
 * Write a string to serial port.
 */
void usart_write(char *str);

/*
 * Initialization routine for serial port.
 */
void init_usart();

/*
 * Write a byte to serial port
 */
void usart_sendbyte(uint8_t u8Data);

#endif
```

H Källkod

`numpres.c` är huvudprogrammet.

H.1 numpres.c

```
#include "global.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <avr/eeprom.h>
#include <avr/interrupt.h>
#include <avr/io.h>
#include <avr/sleep.h>

#include "buttons.h"
#include "clock.h"
#include "diodes.h"
#include "dtmf.h"
#include "eeprom.h"
#include "lcd.h"
#include "usart.h"


/* Menu structure. Actual strings stored in EEPROM */
void *main_menu[5] = {STR_PHONEBOOK, STR_BLOCKED_NBRS,
                      STR_DEL_INCOMING, STR_DEL_OUTGOING,
                      STR_SET_DATE_TIME};

void **menu;                // Pointer to chosen menu
char menu_size;             // Size of chosen menu
char menu_choice;           // Chosen menu item
int next_free_call;         // Address to next free call
int next_free_person;       // Address to next free person
char nbr_calls;             // #calls in EEPROM
int nbr_persons;            // #persons in EEPROM
char countdown_incoming;    // #0.5 s incoming calls are displayed
char countdown_menu;        // #0.5 s before back_to_menu()
char scroll[50];            // Text to scroll on LCD line 1
char scroll_pos;            // Position in "scroll"
char call_choice;           // Index of chosen call
char show_time;             // Whether the clock is displayed
char setup_state;           // Which date/time component is selected
char yes_no;                // Menu choice yes(1)/no(0)

/*
 * Get pointer to the string represented by "*addr".
 */
char *str(const char *addr)
{
    static char eeprom_string[26];
```

```
    /* Copy the wanted string (pointed to by "*addr") from internal EEPROM
     * to the string "eeprom_string" */
    eeprom_read_block(eeprom_string, addr, sizeof(eeprom_string));

    return eeprom_string;
}

/*
 * Parse the command and arguments stored in USART buffer
 * (usart_ibuf) and take proper action.
 */
void parse_command()
{
    char command[5];
    char arg[2][32];
    int n = 0;
    struct person_t person;
    int res;
    int found;

    /* Place the first string in command, the second in arg[0],
     * and the rest in arg[1]. Separation by space. */
    res = sscanf(usart_ibuf, str(STR_S_S_N), command, arg[0], &n);
    strncpy(arg[1], &usart_ibuf[n], strlen(usart_ibuf)-n+1);
    arg[1][sizeof(arg[1])-1] = 0;

    if (res) {

        /* "AP" command adds a person in EEPROM.
         * Overwrites an entry if number matches.
         * Syntax: AP [number] [name]
         * Answer: TOO FEW ARGS | PERSON ADDED | PERSON OVERWRITTEN */
        if (strncmp(command, str(STR_AP), sizeof(command)) == 0) {
            if ((res < 2) | (strlen(arg[1]) == 0)) {
                usart_write(str(STR_TOO_FEW_ARGS));
                usart_write(str(STR_R_N));
            } else {
                person.flags = PERSON_NOT_EMPTY;
                strncpy(person.number, arg[0], 16);
                strncpy(person.name, arg[1], 30);
                found = find_number(person.number, 0, 0);
                if (found == -1) {
                    eeprom_from_mem(&person, sizeof(person),
                                    next_free_person);
                    next_free_person += sizeof(person);
                    nbr_persons++;
                    usart_write(str(STR_PERSON_ADDED));
                    usart_write(str(STR_R_N));
                } else {
                    eeprom_from_mem(&person, sizeof(person), found);
                    usart_write(str(STR_PERSON_OVERWRITTEN));
                    usart_write(str(STR_R_N));
```

```
        }
    }

/* "DP" command deletes a person in EEPROM.
 * Syntax: DP [number]
 * Answer: TOO FEW ARGS | DELETED | NOT FOUND */
} else if (strncmp(command, str(STR_DP),
           sizeof(command)) == 0) {
    if (res < 2) {
        usart_write(str(STR_TOO_FEW_ARGS));
        usart_write(str(STR_R_N));
    } else {
        found = find_number(arg[0], 0, 0);
        if (found != -1) {
            eeprom_write_dreg(found, 0);
            nbr_persons--;
            usart_write(str(STR_DELETED));
            usart_write(str(STR_R_N));
        } else {
            usart_write(str(STR_NOT_FOUND));
            usart_write(str(STR_R_N));
        }
    }

/* "QP" command (query person) searches for a number in EEPROM.
 * Syntax: QP [number]
 * Answer: TOO FEW ARGS | NOT FOUND | P [number] [name] */
} else if (strncmp(command, str(STR_QP),
           sizeof(command)) == 0) {
    if (res < 2) {
        usart_write(str(STR_TOO_FEW_ARGS));
        usart_write(str(STR_R_N));
    } else {
        found = find_number(arg[0], 0, arg[1]);
        if (found != -1) {
            usart_sendbyte('P');
            usart_sendbyte(' ');
            usart_write(arg[0]);
            usart_sendbyte(' ');
            usart_write(arg[1]);
            usart_write(str(STR_R_N));
        } else {
            usart_write(str(STR_NOT_FOUND));
            usart_write(str(STR_R_N));
        }
    }

/* Command unknown.
 * Answer: UNKNOWN: [command] */
} else {
    snprintf(arg[0], sizeof(arg[0]), str(STR_UNKN_S_R_N),
             command);
    usart_write(arg[0]);
```

```
        }
    } else {
        usart_write(str(STR_PARSE_ERROR));
    }
}

/*
 * Display the menu.
 */
void display_menu()
{
    int i;
    char buf[17];

    lcd_wait();
    lcd_command(LCD_CLEAR);

    if (menu_choice < menu_size) {

        /* Print the chosen menu item on first LCD line */
        snprintf(buf, sizeof(buf), "%d %-13s\176", menu_choice+1,
                 str(menu[(int)menu_choice]));
        buf[sizeof(buf)-1] = 0;
        lcd_write(buf);

        /* Print next menu item on second LCD line,
         * or "Exit" if end of menu */
        lcd_setaddr(64);
        if (menu_size > (menu_choice+1)) {
            snprintf(buf, sizeof(buf), "%d %-13s", menu_choice+2,
                     str(menu[(int)menu_choice+1]));
            buf[sizeof(buf)-1] = 0;
            lcd_write(buf);
        } else {
            lcd_write(str(STR_ARROW_EXIT));
        }
    } else {
        /* End of menu - print exit choice and delimiter ("---"...) */
        lcd_write(str(STR_ARROW_EXIT));
        lcd_setaddr(64);
        lcd_putchar(' ');
        lcd_putchar(' ');
        for (i=0; i<13; i++) {
            lcd_putchar('-');
        }
    }
}

/*
 * Step one item upwards in the menu.
 */
void menu_up()
{
```

```
    if (menu_choice > 0) {
        menu_choice--;
    } else {
        menu_choice = menu_size;
    }
    display_menu();
}

/*
 * Step one item downwards in the menu.
 */
void menu_down()
{
    menu_choice = (menu_choice + 1) % (menu_size + 1);
    display_menu();
}

/*
 * Print a "YES"/"NO" choice at right in LCD, and indicate
 * selected choice ("yes_no").
 */
void print_yes_no()
{
    lcd_setaddr(13);
    lcd_write(str(STR_NO));

    lcd_setaddr(77);
    lcd_write(str(STR_YES));

    lcd_setaddr(12+(1-yes_no)*64);
    lcd_putchar(' ');

    lcd_setaddr(12+yes_no*64);
    lcd_putchar('\176');

    lcd_command(LCD_CURSOR_ON);
    lcd_setaddr(12+yes_no*64);
}

/*
 * Change yes to no and vice versa in "YES"/"NO" choice.
 */
void change_yes_no()
{
    yes_no = (yes_no) ? 0 : 1;
    print_yes_no();
}

/*
 * Delete all calls from EEPROM.
 */
void delete_incoming()
{
```

```
    int addr;

    if (nbr_calls) {
        addr = next_free_call;
        do {
            addr -= sizeof(struct call_t);
            if ((eeprom_read_dreg(addr) &
                (CALL_NOT_EMPTY | CALL_INCOMING)) ==
                (CALL_NOT_EMPTY | CALL_INCOMING)) {

                eeprom_write_dreg(addr, 0);
            }
        } while (addr >= sizeof(struct call_t));
        memstat();
    }
}

/*
 * Go back to menu. Restore button actions and print the menu.
 */
void back_to_menu()
{
    lcd_command(LCD_CURSOR_OFF);
    set_up_function(menu_up);
    set_down_function(menu_down);
    set_left_function(exit_menu);
    set_right_function(menu_action);

    display_menu();
}

/*
 * Do what user wants (delete calls / back to menu).
 */
void delete_incoming_action()
{
    lcd_command(LCD_CURSOR_OFF);
    if (yes_no) {
        /* Print "WAIT..." */
        lcd_command(LCD_CLEAR);
        lcd_write(str(STR_WAIT));

        /* Disable buttons */
        set_up_function(0);
        set_down_function(0);
        set_left_function(0);
        set_right_function(0);

        /* Call actual delete function */
        delete_incoming();

        /* Print "DONE" */
        lcd_command(LCD_CLEAR);
```

```
        lcd_write(str(STR_DONE));

        /* Display the message for two seoonds or until
         * button is pressed */
        countdown_menu = 4;
        set_up_function(back_to_menu);
        set_down_function(back_to_menu);
        set_left_function(back_to_menu);
        set_right_function(back_to_menu);
    } else {
        back_to_menu();
    }
}

/*
 * Handle the users menu action.
 */
void menu_action()
{
    if (menu == main_menu) {
        switch (menu_choice) {

            /* Delete all incoming calls */
            case 2:
                /* Ask user for confirmation */
                lcd_command(LCD_CLEAR);
                lcd_write(str(STR_DELETE_ALL));
                lcd_setaddr(64);
                lcd_write(str(STR_INCOMING));

                print_yes_no();

                yes_no = 0;
                set_up_function(change_yes_no);
                set_down_function(change_yes_no);
                set_left_function(back_to_menu);
                set_right_function(delete_incoming_action);
                break;

            /* Run time setup */
            case 4:
                setup_time(1);

                /* Go back to the menu */
                lcd_command(LCD_CURSOR_OFF);
                back_to_menu();
                break;

            /* Exit the menu */
            case 5:
                exit_menu();
                break;
        }
```

```
    }
}

/*
 * Show the menu.
 */
void enter_menu()
{
    set_show_time(0);

    menu = main_menu;
    menu_size = sizeof(main_menu)/2;
    menu_choice = 0;

    display_menu();

    set_up_function(menu_up);
    set_down_function(menu_down);
    set_left_function(exit_menu);
    set_right_function(menu_action);
}

/*
 * Exit the menu and display the clock.
 */
void exit_menu()
{
    /* Disable timer2 interrupt used by the scroller */
    TIMSK &= ~(1<<OCIE2);

    set_up_function(show_incoming);
    set_down_function(0);
    set_left_function(0);
    set_right_function(enter_menu);

    /* Clear display and start displaying clock */
    set_show_time(1);

    /* Display the clock immediately,
     * without waiting for the next second */
    lcd_command(LCD_CURSOR_OFF);
    display_clock();
}

/*
 * Show incoming call on LCD and save it to EEPROM.
 */
void incoming_call(const char *infocode, const char *number)
{
    struct call_t call;
    char buf[9];
    char buf2[50];
    char name[30];
```

```
int found;

call.flags = CALL_NOT_EMPTY | CALL_INCOMING | CALL_MISSED;
call.time = now;

/* Hidden or unknown number */
if (infocode != NULL) {
    strncpy(call.number, "", 16);
    if (strncmp(infocode, "00", 3) == 0) {
        call.flags |= CALL_UNKNOWN;
    } else if (strncmp(infocode, "10", 3) == 0) {
        call.flags |= CALL_HIDDEN;
    }

/* Known number */
} else if (number != NULL) {
    strncpy(call.number, number, 16);
    call.number[15] = 0;
}

set_show_time(0);

set_up_function(exit_menu);
set_down_function(exit_menu);
set_left_function(exit_menu);
set_right_function(exit_menu);

tm = localtime(&now);
countdown_incoming = 20;

/* Look for the number in the phonebook */
found = find_number(call.number, 0, name);

/* Print call information to USART */
usart_write("IC ");
usart_write(call.number);
if (found != -1) {
    usart_sendbyte(' ');
    usart_write(name);
}
usart_write(str(STR_R_N));

/* Treat hidden and unknown numbers */
if (call.flags & CALL_UNKNOWN) {
    strcpy(call.number, str(STR_UNKNOWN));
} else if (call.flags & CALL_HIDDEN) {
    strcpy(call.number, str(STR_HIDDEN));
}

/* Format time */
snprintf(buf, sizeof(buf), str(STR_TIME),
         tm->tm_hour, tm->tm_min, tm->tm_sec);
```

```
    /* Display the call on LCD */
    if (found == -1) {
        snprintf(buf2, sizeof(buf2), "%s", call.number);
    } else {
        snprintf(buf2, sizeof(buf2), "%s <%s>", name), call.number;
    }
    start_scroll(buf2, buf);

    /* Save the call in EEPROM */
    save_call(&call);
}

/*
 * Loop through EEPROM, count items and initialize variables.
 * Should be called on start-up before accessing calls or persons.
 */
void memstat()
{
    int addr;
    int last_used;
    int used;
    char flags;

    /* Check call list */
    last_used = -sizeof(struct call_t);
    used = 0;
    for (addr=0; addr<START_PERSON; addr+=sizeof(struct call_t)) {
        flags = eeprom_read_dreg(addr);
        if (flags & CALL_NOT_EMPTY) {
            last_used = addr;
            used++;
        }
    }
    next_free_call = last_used + sizeof(struct call_t);
    nbr_calls = used;

    /* Check person list */
    last_used = START_PERSON - sizeof(struct person_t);
    used = 0;
    for (addr=START_PERSON; addr<END_MEM;
                            addr+=sizeof(struct person_t)) {
        flags = eeprom_read_dreg(addr);
        if (flags & PERSON_NOT_EMPTY) {
            last_used = addr;
            used++;
        }
    }
    next_free_person = last_used + sizeof(struct person_t);
    nbr_persons = used;
}

/*
 * Save the call "*call" to EEPROM.
```

```
 */
void save_call(const struct call_t *call)
{
    if (call) {
        if (nbr_calls < MAX_CALLS) {
            if (next_free_call + sizeof(struct call_t) < START_PERSON) {
                eeprom_from_mem(call, sizeof(struct call_t),
                                next_free_call);
                next_free_call += sizeof(struct call_t);
                nbr_calls++;
            } else {
                // Defrag?
            }
        } else {
            // Delete first call?
        }
    }
}

/*
 * Perform one step of the scrolling.
 */
void do_scroll()
{
    char scrollbuf[17];
    char startpos;
    int j;

    /* Compute scroll offset */
    if (scroll_pos < 16) {
        startpos = 0;
    } else {
        startpos = scroll_pos - 16;
    }
    strncpy(scrollbuf, &scroll[(int)startpos], sizeof(scrollbuf));
    for (j=strlen(scrollbuf); j<16; j++) {
        scrollbuf[j] = ' ';
    }
    scrollbuf[16] = 0;

    lcd_setaddr(0);

    /* Print spaces if needed */
    if (16 - scroll_pos) {
        for (j=0; j<(16-scroll_pos); j++) {
            lcd_putchar(' ');
        }
    }

    /* Print the scrolling line on LCD */
    lcd_write(scrollbuf);

    scroll_pos++;
```

```
    if (scroll_pos == strlen(scroll)+16) {
       scroll_pos = 0;
    }
}

/*
 * Interrupt routine for timer2, used by the scroller.
 */
SIGNAL(SIG_OUTPUT_COMPARE2)
{
    /* Reset the timer */
    TCNT2 = 0;
    do_scroll();
}

/*
 * Initialize the scroller.
 */
void start_scroll(const char *line1, const char *line2)
{
    strncpy(scroll, line1, 50);
    scroll_pos = 0;

    /* Print first line if no scrolling needed */
    if (strlen(line1) <= 15) {
        lcd_setaddr(0);
        lcd_write(line1);
    }

    /* Print second line */
    lcd_setaddr(64);
    lcd_write(line2);

    /* Initialize timer2 to perform the scrolling */
    if (strlen(line1) > 16) {
        /* Prescale 1024 */
        TCCR2 |= (1<<CS20)|(1<<CS21)|(1<<CS22);

        /* Scroll every 150*1024 clock cycle */
        OCR2 = 150;

        /* Reset timer2 */
        TCNT2 = 0;

        /* Enable timer2 interrupt */
        TIMSK |= (1<<OCIE2);
    }
}

/*
 * Find person with number "*number" in EEPROM.
 * Save found information to "*flags" and "*name".
 * Return the persons address in EEPROM.
```

```
 */
int find_number(const char *number, char *flags, char *name)
{
    char found = 0;
    char count = 0;
    struct person_t person;
    int addr;

    addr = next_free_person;

    /* Loop through EEPROM until a person is found,
     * or all records evaluated */
    do {
        addr -= sizeof(struct person_t);
        if (eeprom_read_dreg(addr) & PERSON_NOT_EMPTY) {
            count++;

            /* Read all information about the person to process */
            eeprom_to_mem(addr, sizeof(person), &person);

            if (strncmp(person.number, number,
                        sizeof(person.number)) == 0) {
                found = 1;

                /* Save the persons flags if pointer given */
                if (flags) {
                    *flags = person.flags;
                }

                /* Save the persons name if pointer given */
                if (name) {
                    strncpy(name, person.name, sizeof(person.name));
                }
            }
        }
    } while ((!found) && (addr > START_PERSON) &&
             (count < nbr_persons));

    if (found) {
        return addr;
    } else {
        return -1;
    }
}

/*
 * Display the call chosen by "call_choice".
 */
void display_call()
{
    int addr;
    int found = 0;
    struct call_t call;
```

```
    char buf[49];
    char buf2[17];

    addr = next_free_call;

    /* Let "addr" point to call number "call_choice" */
    do {
        addr -= sizeof(struct call_t);
        if (eeprom_read_dreg(addr) & CALL_NOT_EMPTY) {
            found++;
        }
    } while (found < (nbr_calls - call_choice + 1));

    /* Read information about the call */
    eeprom_to_mem(addr, sizeof(call), &call);

    /* Look for number in person list */
    found = find_number(call.number, 0, buf);

    /* Prepare strings for presentation */
    if (found != -1) {
        snprintf(&buf[strlen(buf)], sizeof(buf)-strlen(buf),
                 str(STR_LT_S_GT), call.number);
    } else {
        strncpy(buf, call.number, sizeof(buf));
    }
    tm = localtime(&(call.time));
    snprintf(buf2, sizeof(buf2), str(STR_TIME_DATE), call_choice,
             tm->tm_hour, tm->tm_min, tm->tm_mday,
             __month[(int)(tm->tm_mon)]);
    buf[sizeof(buf)-1] = 0;
    buf2[sizeof(buf2)-1] = 0;

    /* Print information about the call on LCD */
    lcd_command(LCD_CLEAR);
    start_scroll(buf, buf2);
}

/*
 * Step one item upwards in the call list
 */
void call_up()
{
    int i;

    /* Disable timer2 interrupt */
    TIMSK &= ~(1<<OCIE2);

    /* Top of list. Print delimiter ("---"...) */
    if (call_choice <= 1) {
        call_choice = 0;
        lcd_command(LCD_CLEAR);
        for (i=0; i<16; i++) {
```

```
            lcd_putchar('-');
        }

    /* Display the new call */
    } else {
        call_choice--;
        display_call();
    }
}

/*
 * Step one item downwards in the call list
 */
void call_down()
{
    int i;

    /* Disable timer2 interrupt */
    TIMSK &= ~(1<<OCIE2);

    call_choice++;

    /* Bottom of list. Print delimiter ("---"...) */
    if (call_choice > nbr_calls) {
        call_choice = nbr_calls + 1;
        lcd_command(LCD_CLEAR);
        lcd_setaddr(64);
        for (i=0; i<16; i++) {
            lcd_putchar('-');
        }

    /* Display the new call */
    } else {
        display_call();
    }
}

/*
 * Show incoming calls.
 */
void show_incoming()
{
    /* Stop displaying clock */
    set_show_time(0);

    /* Start with the latest */
    call_choice = nbr_calls;

    if (nbr_calls > 0) {
        /* Display the call */
        display_call();

        /* Set button functions for navigating in the list */
```

```
        set_up_function(call_up);
        set_down_function(call_down);
        set_right_function(0);

    } else {
        /* Show "No incoming calls" for two seconds,
         * or until button pressed */

        set_up_function(exit_menu);
        set_down_function(exit_menu);
        set_left_function(exit_menu);
        set_right_function(exit_menu);

        countdown_incoming = 4;

        lcd_command(LCD_CLEAR);
        lcd_setaddr(4);
        lcd_write(str(STR_NO_INCOM));
        lcd_setaddr(68);
        lcd_write(str(STR_ING_CALLS));
    }

    set_left_function(exit_menu);
}

int main(void)
{
    /* Enable sleep for power saving */
    MCUCR |= (1<<SE);

    /* Initialize necessary parts of the CID */
    init_buttons();
    init_diodes();
    init_usart();
    init_dtmf();
    init_lcd();
    init_clock();
    init_eeprom();

    /* Inilialize EEPROM variables */
    memstat();

    /* Display interface for setting date and time,
     * without ability to exit */
    setup_time(0);

    /* Display the clock */
    lcd_command(LCD_CURSOR_OFF);
    display_clock();
    set_show_time(1);

    /* Set button functions */
    set_up_function(show_incoming);
```

```
    set_down_function(0);
    set_left_function(0);
    set_right_function(enter_menu);

    while (1) {
        /* Sleep until interrupt occurs */
        sleep_mode();
    }

    return 0;
}
```

H.2 buttons.c

```
#include "buttons.h"
#include "global.h"
#include <stdlib.h>
#include <avr/interrupt.h>
#include <avr/io.h>

char btn_pressed;            // The button pressed
char count;                    // Countdown from minimal time pressed

void (*up_function)();
void (*down_function)();
void (*left_function)();
void (*right_function)();

/*
 * Interrupt handler for buttons.
 * Reset timer0 for later check which buttons are pressed.
 */
SIGNAL(SIG_INTERRUPT1)
{
    btn_pressed = get_pressed_key();

    /* Reset timer */
    TCNT0 = 0;

    /* Enable timer0 interrupt */
    TIMSK |= (1<<OCIE0);

    count = 0;
}

/*
 * Interrupt handler for getting rid of button bounces, using timer0.
 */
SIGNAL(SIG_OUTPUT_COMPARE0)
{
    char key;

    /* Reset timer0 */
```

```
    TCNT0 = 0;

    key = get_pressed_key();

    if (key == btn_pressed) {
        count++;
    } else {
        count = 0;
        btn_pressed = key;
    }

    /* Consider the button pressed after three timer interrupts */
    if (count > 2) {
        count = 0;

        /* Disable timer0 interrupt */
        TIMSK &= ~(1<<OCIE0);

        if (key_pressed(KEY_UP) && up_function > 0) {
            up_function();
        }
        if (key_pressed(KEY_DOWN) && down_function > 0) {
            down_function();
        }
        if (key_pressed(KEY_LEFT) && left_function > 0) {
            left_function();
        }
        if (key_pressed(KEY_RIGHT) && right_function > 0) {
            right_function();
        }
        btn_pressed = 0;
    }
}

/*
 * Set the function to be called when button UP is pressed.
 */
void set_up_function(void (*func)())
{
    up_function = func;
}

/*
 * Set the function to be called when button DOWN is pressed.
 */
void set_down_function(void (*func)())
{
    down_function = func;
}

/*
 * Set the function to be called when button LEFT is pressed.
 */
```

```
void set_left_function(void (*func)())
{
    left_function = func;
}

/*
 * Set the function to be called when button RIGHT is pressed.
 */
void set_right_function(void (*func)())
{
    right_function = func;
}

/*
 * Initialization routine for buttons.
 */
void init_buttons()
{
    count = 0;

    /* Disable pull-ups */
    PORTD &= ~(1<<PD4 | 1<<PD5 | 1<<PD6 | 1<<PD7);

    /* Set input mode */
    DDRD &= ~(1<<DDD4 | 1<<DDD5 | 1<<DDD6 |1<<DDD7);

    /* Prescale 256 */
    TCCR0 |= (1<<CS02);

    /* Time between timer interrupts: 10*256 clock cycles */
    OCR0 = 10;

    /* INT1 on rising edge */
    MCUCR |= (1<<ISC11)|(1<<ISC10);

    /* INT1 enable */
    GICR |= (1<<INT1);

    /* Global interrupt enable */
    sei();
}

/*
 * Get which keys are pressed
 */
char get_pressed_key()
{
    char keys = 0;

    if (PIND & KEY_BIT_UP)
        keys |= KEY_UP;
    if (PIND & KEY_BIT_DOWN)
        keys |= KEY_DOWN;
```

```
    if (PIND & KEY_BIT_LEFT)
        keys |= KEY_LEFT;
    if (PIND & KEY_BIT_RIGHT)
        keys |= KEY_RIGHT;

    return keys;
}

/*
 * Check if the particular key "key" is pressed
 */
char key_pressed(char key)
{
    return ((btn_pressed & key) ? 1 : 0);
}
```

H.3 clock.c

```
#include "clock.h"
#include "global.h"
#include <stdio.h>
#include <avr/interrupt.h>
#include <avr/io.h>
#include <avr/sleep.h>
#include "buttons.h"
#include "lcd.h"
#include "time.h"

/* Stores the date and time in UNIX timestamp format.
 * Initializes to this value at power on. */
time_t now = 1165277800;

struct tm *tm, new_time;    // Structs for converting to other formats
char multiple;                // Counter increases every half a second
char setup_exitable;        // Whether the time setup can be quitted
volatile char time_set;        // Whether the date and time have been set

/* Store the current day of year, in order to
 * print the date in the clock only once a day */
char yday;

/* Cursor positions on LCD when setting up date and time */
char cursor_pos[8] = {67, 70, 73, 75, 65, 68, 71, 76};

/*
 * Clock functionality. Timer1 fires twice a second.
 */
SIGNAL(SIG_OUTPUT_COMPARE1A){
    if (multiple++%2 == 0) {
        now++;

        /* Update clock in LCD if it is displayed */
        if (show_time) {
```

```
            display_clock();
        }
    }

    /* Count down before exiting menu */
    if (countdown_incoming) {
        countdown_incoming--;
        if (countdown_incoming == 0) {
            exit_menu();
        }
    }

    /* Count down before going back to menu */
    if (countdown_menu) {
        countdown_menu--;
        if (countdown_menu == 0) {
            back_to_menu();
        }
    }
}

/*
 * Display the clock in the LCD.
 */
void display_clock()
{
    char timebuf[16];

    tm = localtime(&now);

    /* Only update second line once a day if already displayed */
    if (tm->tm_yday != yday) {
        yday = tm->tm_yday;
        snprintf(timebuf, sizeof(timebuf), str(STR_DATE),
                 __day[tm->tm_wday], tm->tm_mday, __month[tm->tm_mon],
                 1900+tm->tm_year);
        timebuf[sizeof(timebuf)-1] = 0;
        lcd_setaddr(64);
        lcd_write(timebuf);
    }

    /* Display the time on first LCD line */
    lcd_setaddr(4);
    snprintf(timebuf, sizeof(timebuf), str(STR_TIME), tm->tm_hour,
             tm->tm_min, tm->tm_sec);
    timebuf[sizeof(timebuf)-1] = 0;
    lcd_write(timebuf);
}

/*
 * Check whether the date and time are correct.
 */
void setup_check()
```

```
{
    if (new_time.tm_year > 138) {
        new_time.tm_year = 70;
    } else if (new_time.tm_year < 70) {
        new_time.tm_year = 138;
    }
    if (new_time.tm_mon > 240) {
        new_time.tm_mon = 11;
    } else if (new_time.tm_mon > 11) {
        new_time.tm_mon = 0;
    }
    if (new_time.tm_mday == 0) {
        if (new_time.tm_mon == 1) {
            new_time.tm_mday = (new_time.tm_year % 4 == 0) ? 29 : 28;
        } else {
            new_time.tm_mday = monthDays[new_time.tm_mon];
        }
    } else if (new_time.tm_mday > monthDays[new_time.tm_mon]) {
        if (new_time.tm_mon == 1) {
            if (new_time.tm_mday >
                            ((new_time.tm_year % 4 == 0) ? 29 : 28)) {
                new_time.tm_mday = 1;
            }
        } else {
            new_time.tm_mday = 1;
        }
    }
    if (new_time.tm_hour > 240) {
        new_time.tm_hour = 23;
    } else if (new_time.tm_hour > 23) {
        new_time.tm_hour = 0;
    }
    if (new_time.tm_min > 240) {
        new_time.tm_min = 59;
    } else if (new_time.tm_min > 59) {
        new_time.tm_min = 0;
    }
    if (new_time.tm_sec > 240) {
        new_time.tm_sec = 59;
    } else if (new_time.tm_sec > 59) {
        new_time.tm_sec = 0;
    }
}

/*
 * Increment the selected position in date/time.
 */
void set_time_up()
{
    switch (setup_state) {
        case 0:
            new_time.tm_year++;
            break;
```

```
        case 1:
            new_time.tm_mon++;
            break;
        case 2:
            new_time.tm_mday++;
            break;
        case 4:
            new_time.tm_hour++;
            break;
        case 5:
            new_time.tm_min++;
            break;
        case 6:
            new_time.tm_sec++;
            break;
    }
    setup_check();
    setup_refresh();
}

/*
 * Decrement the selected position in date/time.
 */
void set_time_down()
{
    switch (setup_state) {
        case 0:
            new_time.tm_year--;
            break;
        case 1:
            new_time.tm_mon--;
            break;
        case 2:
            new_time.tm_mday--;
            break;
        case 4:
            new_time.tm_hour--;
            break;
        case 5:
            new_time.tm_min--;
            break;
        case 6:
            new_time.tm_sec--;
            break;
    }
    setup_check();
    setup_refresh();
}

/*
 * Move selected position in date/time to the left.
 */
void set_time_left()
```

```
{
    if (setup_exitable && setup_state == 0) {
        time_set = 1;
    } else if (setup_state > 0) {
        setup_state--;
    }
    setup_refresh();
}

/*
 * Move selected position in date/time to the right.
 */
void set_time_right()
{
    if (setup_state < 7) {
        setup_state++;
    } else {
        /* Set time */
        yday = 0;
        TCNT1 = 0;
        now = mktime(&new_time);
        time_set = 1;
    }
    setup_refresh();
}

/*
 * Refresh the date/time setup on LCD.
 */
void setup_refresh()
{
    char buf[17];
    char cursor;

    lcd_wait();
    lcd_command(LCD_CLEAR);

    /* Date mode */
    if (setup_state < 4) {
        lcd_write(str(STR_SET_DATE));
        if (setup_state == 0 && setup_exitable) {
            lcd_write(str(STR_ARROW_EXIT2));
        }
        lcd_setaddr(40);
        snprintf(buf, sizeof(buf), str(STR_SETUP_DATE),
                 1900+(new_time.tm_year), 1+new_time.tm_mon,
                 new_time.tm_mday);
        lcd_write(buf);
        lcd_write(str(STR_ARROW_TIME));
        cursor = cursor_pos[(int)setup_state];
        lcd_setaddr(cursor);

    /* Time mode */
```

```
    } else {
        lcd_write(str(STR_SET_TIME));
        lcd_setaddr(40);
        snprintf(buf, sizeof(buf), str(STR_TIME), new_time.tm_hour,
                 new_time.tm_min, new_time.tm_sec);
        lcd_write(buf);
        lcd_write(str(STR_ARROW_SET));
        cursor = cursor_pos[(int)setup_state];
        lcd_setaddr(cursor);
    }
}

/*
 * Setup date and time.
 * "exitable" controls whether the dialog can be quitted.
 */
void setup_time(char exitable)
{
    setup_state = 0;
    time_set = 0;
    setup_exitable = exitable;

    /* Set the setup-time to the current time */
    clone_tm(localtime(&now), &new_time);

    /* Show the setup dialog on LCD */
    lcd_command(LCD_CURSOR_ON);
    setup_refresh();

    /* Set button functions */
    set_left_function(set_time_left);
    set_right_function(set_time_right);
    set_up_function(set_time_up);
    set_down_function(set_time_down);

    /* Enable new interrupts */
    sei();

    /* Wait until date/time setup dialog finishes */
    while (time_set != 1);
}

/*
 * Copy a tm struct to another.
 */
void clone_tm(struct tm *src, struct tm *dst)
{
    dst->tm_sec = src->tm_sec;
    dst->tm_min = src->tm_min;
    dst->tm_hour = src->tm_hour;
    dst->tm_mday = src->tm_mday;
    dst->tm_mon = src->tm_mon;
    dst->tm_year = src->tm_year;
```

```
    dst->tm_wday = src->tm_wday;
    dst->tm_yday = src->tm_yday;
    dst->tm_isdst = src->tm_isdst;
    dst->tm_hundredth = src->tm_hundredth;
}

/*
 * Enable(1)/disable(0) displaying of clock.
 */
void set_show_time(char state)
{
    show_time = state;

    /* Clear the display. */
    lcd_wait();
    lcd_command(LCD_CLEAR);

    /* Remark: the clock will be displayed at next second */
}

/*
 * Initialization routing for the clock.
 */
void init_clock()
{
    show_time = 0;
    yday = 0;

    /* Timer1: prescale 64, CTC mode */
    TCCR1B |= (1<<WGM12)|(1<<CS11)|(1<<CS10);

    /* Calibration done to get two interrupts per second */
    OCR1A = 7936;

    /* Reset timer1 */
    TCNT1 = 0;

    multiple = 0;
    countdown_incoming = 0;
    countdown_menu = 0;

    clone_tm(localtime(&now), &new_time);

    /* Enable timer1 interrupt */
    TIMSK |= (1<<OCIE1A);
}
```

H.4 diodes.c

```
#include "diodes.h"
#include <avr/io.h>

/*
```

```
 * Initialization routine for diodes
 */
void init_diodes()
{
    /* Set diode pins for output */
    DDRC = (1<<DDC7)|(1<<DDC6);

    /* Turn off diodes */
    PORTC &= (RED|GREEN);
}
```

H.5 dtmf.c

```
#include "dtmf.h"
#include "global.h"
#include <stdio.h>
#include <avr/interrupt.h>
#include <avr/io.h>
#include <util/delay.h>

char dtmf_initialized;      // Whether the init_dtmf() has been run
char state;                 // 0=idle, 1=caller, 2=status
char infocode[3];           // Buffer for infocode
char number[16];            // Buffer for number
char infocode_count;        // Size of infocode
char number_count;          // Size of number

/* Declare DTMF digits */
char digit[] = {'D', '1', '2', '3',
                '4', '5', '6', '7',
                '8', '9', '0', '*',
                '#', 'A', 'B', 'C'};

/*
 * Read the status of DTMF chip.
 */
char dtmf_read_status()
{
    char status;

    dtmf_deselect();

    /* Data input */
    DDRA &= DATAMASK;

    /* Read status register */
    PORTA = (R_SR) + (PORTA & DTMFMASK);

    dtmf_select();
    status = PINA & ~DATAMASK;
    dtmf_deselect();

    return status;
```

```
}

/*
 * Read data from DTMF chip.
 */
int dtmf_read_data()
{
    int data;

    dtmf_deselect();

    /* Data input */
    DDRA &= DATAMASK;

    /* Read receive data register */
    PORTA = (R_RR) + (PORTA & DTMFMASK);

    dtmf_select();
    data = PINA & ~DATAMASK;
    dtmf_deselect();

    return data;
}

/*
 * Send data to DTMF chip.
 */
int dtmf_send_data(char number)
{
    int status = dtmf_read_status();

    if (status & S_TX) {
        PORTA = number + (PORTA & DATAMASK);
    }

    return 0;
}

/*
 * Interrupt handler for DTMF chip.
 */
SIGNAL(SIG_INTERRUPT0)
{
    int status = dtmf_read_status();
    char data;

    if (status & S_RX) {
        /* Data to fetch in receive register */

        data = digit[dtmf_read_data()];
        if (data == 'B' && state != 2) {
            /* Infocode expected to follow */
            state = 2;
```

```
        } else if (data == 'D' && state != 1) {
            /* Number expected to follow */
            state = 1;
        } else if (data >= '0' && data <= '9' && state == 1 &&
                   number_count < 15) {
            /* Buffer number */
            number[(int)number_count++] = data;
        } else if (data >= '0' && data <= '9' && state == 2 &&
                   infocode_count < 2) {
            /* Buffer infocode */
            infocode[(int)infocode_count++] = data;
        } else if (data == 'C' && state > 0) {
            /* End of number/infocode. Process the information. */

            if (state == 1) {
                number[(int)number_count] = 0;
                incoming_call(0, number);
            } else if (state == 2) {
                infocode[(int)infocode_count] = 0;
                incoming_call(infocode, 0);
            }

            /* Reset buffers and get ready for next call */
            goto CLEAR_DTMF;
        } else {
CLEAR_DTMF:
            /* Erroneous behaviour, clear buffers and jump to state 0 */
            state = 0;
            infocode_count = 0;
            number_count = 0;
        }
    }

    /* Transmitter ready for new data */
    if (status & S_TX) {
    }
}

/*
 * Initialization routine for DTMF chip.
 */
void init_dtmf()
{
    dtmf_initialized = 1;

    /* Set CS to output */
    DDRC |= (1<<DDC1);

    /* Deselect DTMF chip */
    PORTC |= DTMF_CS;

    /* Set pins R/W and RS0 to output */
    DDRA |= DTMF_RW|DTMF_RS0;
```

```
    /* Pull-up interrupt pin */
    PORTD |= (1<<PD2);

    /* Data output */
    DDRA |= ~DATAMASK;

    PORTA = (W_CR) + (PORTA & DTMFMASK);

    dtmf_select();
    dtmf_deselect();
    dtmf_select();
    dtmf_deselect();

    /* Write Control Register A */
    PORTA = (W_CR|RSEL|IRQ) + (PORTA & DTMFMASK);
    dtmf_select();
    dtmf_deselect();

    /* Write Control Register B */
    PORTA = (W_CR) + (PORTA & DTMFMASK);
    dtmf_select();
    dtmf_deselect();

    infocode_count = 0;
    number_count = 0;

    /* INT0 on falling edge */
    MCUCR |= (1<<ISC01);

    /* INT0 enable */
    GICR |= (1<<INT0);

    /* Global interrupt enable */
    sei();
}

/*
 * Select DTMF chip on the bus (enable).
 */
void dtmf_select()
{
    if (!dtmf_initialized) {
        init_dtmf();
    }

    /* Select DTMF chip */
    PORTC &= ~DTMF_CS;

    /* Set R/W and RS0 to output */
    DDRA |= DTMF_RW|DTMF_RS0;
}
```

```
/*
 * Deselect DTMF chip on the bus (disable).
 */
void dtmf_deselect()
{
    if (!dtmf_initialized) {
        init_dtmf();
    }

    /* Deselect DTMF chip */
    PORTC |= DTMF_CS;
}
```

H.6 eeprom.c

```
#include "eeprom.h"
#include "global.h"
#include <avr/io.h>
#include <util/delay.h>

/*
 * Initialization routine for external EEPROM.
 */
void init_eeprom()
{
    spi_init_as_master();
    eeprom_deselect();
    _delay_loop_2(50);
    eeprom_select();
}

/*
 * Initialize the ATmega16 as master.
 */
void spi_init_as_master()
{
    /* Set MOSI, SCK, CS output */
    DDRB |= (1<<DDB7)|(1<<DDB5)|(1<<DDB1);

    /* Pull-up for input */
    PORTB |= (1<<DDB6);

    /* Enable SPI, set master, set clock rate fck/16, send MSB first */
    SPCR = (1<<SPE) | (1<<MSTR) | (1<<SPR0);
}

/*
 * Wait until SPI-transmission is complete.
 */
void spi_wait()
{
    while (!(SPSR & (1<<SPIF)));
}
```

```
/*
 * Transmit data on SPI bus.
 */
void spi_transmit(char cData)
{
    /* Start transmission (place data in spi data reg) */
    SPDR = cData;

    spi_wait();

    SPDR;
}

/*
 * Receive data on SPI bus.
 */
char spi_receive()
{
    SPDR = 0;
    spi_wait();

    /* Read data */
    return SPDR;
}

/*
 * Enable EEPROM chip.
 */
void eeprom_select()
{
    _delay_loop_2(40);

    /* Select the EEPROM by pulling CS' low */
    PORTB &= ~EEPROM_CS;

    _delay_loop_2(40);
}

/*
 * Disable EEPROM chip.
 */
void eeprom_deselect()
{
    _delay_loop_2(40);

    /* Deselect the EEPROM by taking the CS' high */
    PORTB |= EEPROM_CS;

    _delay_loop_2(40);
}

/*
```

```
 * Wait until EEPROM ready.
 */
void eeprom_wait()
{
    while(eeprom_read_sreg() & 0x01);
}

/*
 * Prepare the EEPROM for writing instructions.
 */
void eeprom_write_enable(void)
{
    eeprom_select();

    /* Send write enable instruction */
    spi_transmit(EEPROM_C_WREN);

    eeprom_deselect();
}

/*
 * Write data to status register.
 */
void eeprom_write_sreg(char opcode)
{
    /* Prepare for writing status register */
    eeprom_write_enable();
    eeprom_select();
    spi_transmit(EEPROM_C_WRSR);

    /* Write status register */
    spi_transmit(opcode);

    eeprom_deselect();
}

/*
 * Read data from status register.
 */
char eeprom_read_sreg()
{
    char data;

    /* Prepare for reading status register */
    eeprom_select();
    spi_transmit(EEPROM_C_RDSR);

    /* Read data */
    data = spi_receive();

    eeprom_deselect();

    return data;
```

```
}

/*
 * Read data from data register.
 */
char eeprom_read_dreg(uint16_t addr)
{
    uint8_t buffer;

    /* Prepare for reading */
    eeprom_wait();
    eeprom_select();
    spi_transmit(EEPROM_C_READ);

    /* Send high address part */
    spi_transmit(addr >> 8);

    /* Send low address part */
    spi_transmit(addr & 0xFF);

    /* Read data */
    buffer =  spi_receive();
    eeprom_deselect();

    return buffer;
}

/*
 * Write data to data register.
 */
void eeprom_write_dreg(uint16_t addr, uint8_t data)
{
    /* Prepare for writing */
    eeprom_wait();
    eeprom_write_enable();
    eeprom_select();
    _delay_loop_2(50);
    spi_transmit(EEPROM_C_WRITE);

    /* Send high part */
    spi_transmit(addr >> 8);

    /* Send low part */
    spi_transmit(addr & 0xFF);

    /* Send data */
    spi_transmit(data);

    _delay_loop_2(50);
    eeprom_deselect();
    _delay_loop_2(50);
}
```

```
/*
 * Read "size" bytes from memory (address "*data")
 * and write to EEPROM on address "addr".
 */
void eeprom_from_mem(const void *data, int size, uint16_t addr)
{
    int i;

    for (i=0; i<size; i++) {
        eeprom_write_dreg(addr++, *((char*)data++));
    }
}

/*
 * Read "size" bytes from EEPROM (address "addr")
 * and write to memory on address "*data"
 */
void eeprom_to_mem(uint16_t addr, int size, void *data)
{
    int i;

    for (i=0; i<size; i++) {
        *((char*)data++) = eeprom_read_dreg(addr++);
    }
}
```

H.7 lcd.c

```
#include "lcd.h"
#include "global.h"
#include <avr/io.h>
#include <util/delay.h>
#include "dtmf.h"    // In order to deselect DTMF chip

/*
 * Writes a character to LCD at the current address.
 */
void lcd_putchar(char data)
{
    lcd_wait();
    dtmf_deselect();
    switch (data) {
        case 'å':
            data = 'a';
            break;
        case 'ä':
            data = 0xE1;
            break;
        case 'ö':
            data = 0xEF;
    }

    /* PORTA is output */
```

```
    DDRA = 255;

    /* Put data on bus */
    PORTA = data;

    /* RW low, E low */
    PORTB &= ~(LCD_RW|LCD_E);

    /* RS high, strobe E */
    PORTB |= (LCD_RS|LCD_E);

    _delay_loop_2(2);

    /* RS low again, E low (belongs to strobe) */
    PORTB &= ~(LCD_RS|LCD_E);

    /* Release bus */
    DDRA = 0;
}

/*
 * Get the busy flag and address.
 */
char lcd_getaddr()
{
    char address;

    dtmf_deselect();

    /* PORTA is input */
    DDRA = 0;

    /* RW high, strobe enable */
    PORTB |= (LCD_RW|LCD_E);

    _delay_loop_2(7);

    /* While E is high, get data from LCD */
    address = PINA;

    /* Reset RW to low, E low (for strobe) */
    PORTB &= ~(LCD_RW|LCD_E);

    /* Return address and busy flag */
    return address;
}

/*
 * Set the display's address.
 */
void lcd_setaddr(char address)
{
    dtmf_deselect();
```

```
    lcd_wait();

    /* PORTA is output */
    DDRA = 255;

    /* Put data on bus */
    PORTA = 128 | address;

    /* RW high, strobe enable */
    PORTB = (PORTB & ~LCD_RW) | LCD_E;

    _delay_loop_2(7);

    /* Reset RW to low, E low (for strobe) */
    PORTB &= ~(LCD_RW|LCD_E);
}

/*
 * Wait until the display is ready.
 */
void lcd_wait()
{
    /* Get address and busy flag and loop until busy flag cleared */
    while((lcd_getaddr() & 0x80) == 0x80);
}

/*
 * Tell the display to run an instruction.
 */
void lcd_command(char command)
{
    dtmf_deselect();

    /* PORTA is output */
    DDRA = 255;

    /* Put data on bus */
    PORTA = command;

    /* Reset RS to low, RW to low, E low (for strobe) */
    PORTB &= ~(LCD_RS|LCD_RW|LCD_E);

    /* Strobe enable */
    PORTB |= LCD_E;

    _delay_loop_2(2);

    /* E low (for strobe) */
    PORTB &= ~LCD_E;

    /* Release bus */
    DDRA = 0;
}
```

```
/*
 * Initialization routine for LCD.
 */
void init_lcd()
{
    dtmf_deselect();
    DDRB |= (LCD_RS|LCD_RW|LCD_E);
    PORTB = 0x00;
    _delay_loop_2(0xFFFF);

    /* Tell the LCD that it's used in 8-bit mode 3 times,
     * each with a delay inbetween */
    lcd_command(0x30);
    _delay_loop_2(0xFFFF);
    _delay_loop_2(0xFFFF);
    lcd_command(0x30);
    _delay_loop_2(0xFFFF);
    _delay_loop_2(0xFFFF);
    lcd_command(0x30);
    _delay_loop_2(0xFFFF);
    _delay_loop_2(0xFFFF);

    /* 8 bit interface, 5*7 font, 2 lines */
    lcd_command(0x38);

    /* Display on */
    lcd_wait();
    lcd_command(LCD_CURSOR_OFF);

    /* Clear the display, cursor home */
    lcd_wait();
    lcd_command(LCD_CLEAR);

    /* Cursor auto-increment */
    lcd_wait();
    lcd_command(0x06);
}

/*
 * Write string to display
 */
void lcd_write(const char *dstring)
{
    while(*dstring) {
        lcd_putchar(*dstring++);
    }
}
```

H.8 time.c

```
/*
 * Copyright Droids Corporation, Microb Technology, Eirbot (2005)
```

```
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston,
 * MA 02111-1307 USA
 *
 * Revision : $Id: time.c,v 1.2 2005/02/15 13:01:14 zer0 Exp $
 *
 */

// pompage total, trouvé dans les libs de sdcc

/*-------------------------------------------------------------------------
time.c - stdlib time conversion routines

 Written By - Johan Knol, johan.knol@iduna.nl

 This program is free software; you can redistribute it and/or modify it
 under the terms of the GNU General Public License as published by the
 Free Software Foundation; either version 2, or (at your option) any
 later version.

 This program is distributed in the hope that it will be useful,
 but WITHOUT ANY WARRANTY; without even the implied warranty of
 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 GNU General Public License for more details.

 You should have received a copy of the GNU General Public License
 along with this program; if not, write to the Free Software
 Foundation, 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

 In other words, you are welcome to use, share and improve this program.
 You are forbidden to forbid anyone else to use, share and improve
 what you give them.   Help stamp out software-hoarding!
-------------------------------------------------------------------------*/

#include <stdio.h>
#include "time.h"

// please note that the tm structure has the years since 1900,
// but time returns the seconds since 1970

/* You need some kind of real time clock for the time() function.
```

```
   Either a rtc-chip or some kind of DCF device will do. For TINI, the
   HAVE_RTC is defined in tinibios.h
   If not, the conversion routines still work.
*/

#ifndef HAVE_RTC
unsigned char RtcRead(struct tm *timeptr) {
  // no real time hardware
  //timeptr; // hush the compiler
  return 0;
}
#endif

char monthDays[]={31,28,31,30,31,30,31,31,30,31,30,31};

char *__month[]={"Jan","Feb","Mar","Apr","May","Jun",
                              "Jul","Aug","Sep","Oct","Nov","Dec"};

char *__day[]={"Sun","Mon","Tue","Wed","Thu","Fri","Sat"};

//static char ascTimeBuffer[32];

// validate the tm structure
static void CheckTime(struct tm *timeptr) {
    // we could do some normalization here, e.g.
    // change 40 october to 9 november
    #if !__TIME_UNSIGNED
    if (timeptr->tm_sec<0) timeptr->tm_sec=0;
    if (timeptr->tm_min<0) timeptr->tm_min=0;
    if (timeptr->tm_hour<0) timeptr->tm_hour=0;
    if (timeptr->tm_wday<0) timeptr->tm_wday=0;
    if (timeptr->tm_mon<0) timeptr->tm_mon=0;
    #endif

    if (timeptr->tm_sec>59) timeptr->tm_sec=59;
    if (timeptr->tm_min>59) timeptr->tm_min=59;
    if (timeptr->tm_hour>23) timeptr->tm_hour=23;
    if (timeptr->tm_wday>6) timeptr->tm_wday=6;
    if (timeptr->tm_mday<1) timeptr->tm_mday=1;
    else if (timeptr->tm_mday>31) timeptr->tm_mday=31;
    if (timeptr->tm_mon>11) timeptr->tm_mon=11;
    if (timeptr->tm_year<0) timeptr->tm_year=0;
}

static struct tm lastTime;

/*
 convert calendar time (seconds since 1970) to broken-time
 This only works for dates between 01-01-1970 00:00:00 and
 19-01-2038 03:14:07

 A leap year is ((((year%4)==0) && ((year%100)!=0)) || ((year%400)==0))
 but since we have no fancy years between 1970 and 2038 we can do:
```

```
*/

#define LEAP_YEAR(year) ((year%4)==0)

// forget about timezones for now
struct tm *localtime(time_t *timep) {
    return gmtime(timep);
}

struct tm *gmtime(time_t *timep) {
  unsigned long epoch=*timep;
  unsigned int year;
  unsigned char month, monthLength;
  unsigned long days;

  lastTime.tm_sec=epoch%60;
  epoch/=60; // now it is minutes
  lastTime.tm_min=epoch%60;
  epoch/=60; // now it is hours
  lastTime.tm_hour=epoch%24;
  epoch/=24; // now it is days
  lastTime.tm_wday=(epoch+4)%7;

  year=1970;
  days=0;
  while((days += (LEAP_YEAR(year) ? 366 : 365)) <= epoch) {
    year++;
  }
  lastTime.tm_year=year-1900;

  days -= LEAP_YEAR(year) ? 366 : 365;
  epoch -= days; // now it is days in this year, starting at 0
  lastTime.tm_yday=epoch;

  days=0;
  month=0;
  monthLength=0;
  for (month=0; month<12; month++) {
    if (month==1) { // februari
      if (LEAP_YEAR(year)) {
        monthLength=29;
      } else {
        monthLength=28;
      }
    } else {
      monthLength = monthDays[month];
    }

    if (epoch>=monthLength) {
      epoch-=monthLength;
    } else {
        break;
    }
```

```
  }
  lastTime.tm_mon=month;
  lastTime.tm_mday=epoch+1;

  lastTime.tm_isdst=0;

  return &lastTime;
}

// convert broken time to calendar time (seconds since 1970)
time_t mktime(struct tm *timeptr) {
    int year=timeptr->tm_year+1900, month=timeptr->tm_mon, i;
    long seconds;

    CheckTime(timeptr);

    // seconds from 1970 till 1 jan 00:00:00 this year
    seconds= (year-1970)*(60*60*24L*365);

    // add extra days for leap years
    for (i=1970; i<year; i++) {
        if (LEAP_YEAR(i)) {
            seconds+= 60*60*24L;
        }
    }

    // add days for this year
    for (i=0; i<month; i++) {
      if (i==1 && LEAP_YEAR(year)) {
        seconds+= 60*60*24L*29;
      } else {
        seconds+= 60*60*24L*monthDays[i];
      }
    }

    seconds+= (timeptr->tm_mday-1)*60*60*24L;
    seconds+= timeptr->tm_hour*60*60L;
    seconds+= timeptr->tm_min*60;
    seconds+= timeptr->tm_sec;

    return seconds;
}
```

H.9 usart.c

```
#include "usart.h"
#include "global.h"
#include <stdint.h>
#include <string.h>
#include <avr/interrupt.h>
#include <avr/io.h>

char usart_ibuf[64];      // Receive buffers
```

```
char *usart_in;           // Pointer info the buffer
char usart_icount = 0;    // Counter for the buffer

/*
 * Interrupt handler for RX Complete.
 */
SIGNAL(SIG_USART_RECV)
{
    /* Character sequence for backspace in terminal */
    char backspace[4] = {8, 32, 8, 0};

    if (usart_icount >= 63) {
        /* Buffer full - clear buffer*/
        usart_in = usart_ibuf;
        usart_icount = 0;
    }

    /* Read data */
    *usart_in = UDR;

    usart_icount++;
    if (*usart_in == '\r') {
        /* Command complete - parse command */

        *usart_in = '\0';

        /* Echo */
        usart_write("\r\n");

        parse_command();

        usart_in = usart_ibuf;
        usart_icount = 0;
    } else if (*usart_in == '\n') {
        /* Do nothing! :) */
    } else {
        if (*usart_in == 8) {
            if (usart_icount) {
                usart_icount-=2;
                usart_in--;
                usart_write(backspace);
            }
        } else {
            /* Echo*/
            usart_sendbyte(*usart_in);

            usart_in++;
        }
    }
}

/*
 * Write a string to serial port.
```

```
 */
void usart_write(char *str)
{
    while (*str) {
        usart_sendbyte(*str++);
    }
}

/*
 * Initialization routine for serial port.
 */
void init_usart()
{
    usart_ibuf[0] = '\0';
    usart_in = usart_ibuf;

    /* 4800 baud @ 1 MHz */
    UBRRH = 0;
    UBRRL = 12;

    /* Set frame format to 8 data bits, no parity, 1 stop bit */
    UCSRC = (1<<URSEL)|(1<<UCSZ1)|(1<<UCSZ0);

    /* Enable RX Complete interrupt, receiver and transmitter */
    UCSRB = (1<<RXCIE)|(1<<RXEN)|(1<<TXEN);
}

/*
 * Write a byte to serial port
 */
void usart_sendbyte(uint8_t u8Data)
{
    /* Wait if a byte is being transmitted */
    while((UCSRA&(1<<UDRE)) == 0);

    /* Transmit data */
    UDR = u8Data;
}
```