GSM-alarmmodul

Mats Röösgren

Examensarbete för ingenjörsexamen (YH)

Utbildningsprogrammet för elektroteknik

Vasa 2010

EXAMENSARBETE

Författare:	Mats Röösgren
Utbildningsprogram och ort:	Elektroteknik, Vasa
Inriktningsalternativ/Fördjupning:	Automationsteknik
Handledare:	Dag Björklund

Titel: *GSM-alammodul*

Datum 03.12.2010 Sidantal 36 Bilagor 4

Sammanfattning

Uppgiften var att planera och tillverka en välfungerande prototyp av en GSM-alarmmodul. Alarmmodulen har som uppgift att underrätta alarmets ägare om att ett alarm har utlösts via GSM-förbindelse till ägarens mobiltelefon. Uppgiften löstes med en mikrokontroller som kommunicerar med GSM-modemet i en Siemens mobiltelefon. Till mikrokontrollern kommer alarmsignalen till en digital ingång när ett alarm utlöses. Mikrokontrollern skickar då AT-kommandot för uppringning till GSM-modemet som i sin tur lyder order och ringer upp ägaren. När ägaren till alarmet har noterat att ett alarm har inträffat så ringer denne tillbaka till GSM-alarmmodulen för att kvittera alarmet. Examensarbetet kunde uppdelas i tre huvudkapitel med avseende på själva utförandet; planering av kretsschema, planering av kretskort samt planering och skrivning av programkod. Resultatet blev en väldokumenterad och välfungerande prototyp.

Språk: Svenska Nyckelord: GSM-modem, mikrokontroller, alarm

Förvaras: Tritonia, Vasa vetenskapliga bibliotek

BACHELOR'S THESIS

Author:	Mats Röösgren
Degree Programme:	Electrical engineering, Vaasa
Specialization:	Automation technology
Supervisor:	Dag Björklund

Title: *GSM alarm module*

Date 03.12.2010 Number of pages 36 Appendices 4

Summary

The aim of this thesis was to make a plan for and a prototype of an alarm module with GSM connectivity. The alarm module is to notify the owner of the alarm about a triggered alarm. The notification utilizes a GSM connection and is transmitted to the owner's cell phone. The task was solved by means of a microcontroller which is communicating with the GSM modem inside a Siemens mobile phone. In the event of an alarm the alarm signal enters a digital input on the microcontroller. The microcontroller then sends the AT command for dialing the owner's telephone number to the GSM modem which obeys the command and makes the call. When the owner of the alarm notices that an alarm has occurred, he or she calls the GSM alarm module to clear the alarm. The thesis is divided into three separate main areas considering the practical execution: planning the schematics, planning the PCB and planning and writing software. The result of this thesis is a well documented working prototype.

Language: Swedish Key words: GSM modem, microcontroller, alarm

---

OPINNÄYTETYÖ

Tekijä:	Mats Röösgren
Koulutusohjelma ja paikkakunta:	Elektrotekniikka, Vaasa
Suuntautumisvaihtoehto/Syventävät opinnot:	Automaatiotekniikka
Ohjaajat:	Dag Björklund

Nimike: GSM-hälytinmoduuli

Päivämäärä 03.12.2010 Sivumäärä 36 Liitteet 4

Tiivistelmä

Tehtävä oli suunnitella ja valmistaa toimiva prototyyppi GSM-hälytinmoduulista. Hälytinmoduulin tehtävä on ilmoittaa hälyttimen omistajalle jos hälytys on tapahtunut. Ilmoitus tapahtuu GSM-yhteyden kautta omistajan matkapuhelimeen. Tehtävä ratkaistettiin mikro-ohjaimella joka kommunikoi Siemens matkapuhelimessa olevan GSM-modeemin kanssa. Hälytyksen tapahtuessa mikro-ohjaimen digitaaliseen sisääntuloon tulee hälytyssignaali. Mikro-ohjain lähettää silloin soiton AT-käsky GSM-modeemille, joka vuorostaan tottelee käskyä ja soittaa omistajalle. Kun hälyttimen omistaja on huomannut, että hälytys on tapahtunut, hän soittaa takaisin GSM-hälytysmoduulille kuitatakseen hälytyksen. Opinnäytetyön voi jakaa kolmeen eri osaan suhteessa suoritukseen; piirikaavion suunnittelu, piirilevyn suunnittelu sekä ohjelmakoodin suunnittelu ja kirjoitus. Tulos oli hyvin dokumentoitu ja hyvin toimiva prototyyppi.

Kieli: Ruotsi Avainsanat: GSM-modeemi, mikro-ohjain, hälytin

Innehållsförteckning

1 Inledning

1.1 Beskrivning av uppgiften

Uppgiften är att planera och förverkliga en fungerande prototyp för ett uppringande alarm. Med uppringande alarm menas att alarmet ska ringa upp ägarens mobiltelefon vid alarm. Det uppringande alarmet kan användas i en mängd olika tillämpningar, men den kräver att det finns utrustning som genererar alarmsignalen. Det uppringande alarmets uppgift är bara att ta in en alarmsignal (från övrig utrustning) och meddela ägaren om att ett alarm har utlösts. Det ska alltså inte t.ex. detektera inbrottstjuvar eller rörelse, det överlåts till den externa enhet som genererar alarmsignalen.

Uppgiften kan delas in i följande fyra huvudmoment: planera kretsschema, planera kretskort, skriva programkod samt tillverkning. Prototypen ska planeras och konstrueras så att den uppfyller den utarbetade funktionsspecifikationen i kapitel 1.3.

1.2 Bakgrund till uppgiften

Fasta trådförbindelser har tidigare använts för uppringande alarm, men priset för en sådan linje är högt jämfört med en GSM-anslutning. Vanliga samtal övergår mera och mera till mobiltelefon och den fasta trådlinjen används allt mindre för vanliga samtal. Om man endast använder den fasta trådlinjen för alarmet blir det onödigt kostsamt i drift.

Genom att använda sig av det GSM-modem som finns i en mobiltelefon, i stället för ett dedikerat GSM-modem, kan man tillverka det uppringande alarmet mycket förmånligt, eftersom en mobiltelefon med inbyggt GSM-modem kostar en bråkdel av ett dedikerat GSM-modem.

En ekonomisk konstruktion och användning av prototypen är en viktig del i planeringen och tillverkningen, även om så inte omnämns senare i texten.

1.3 Funktionsspecifikation

1. Alarmet ska kunna ringa upp ett telefonnummer då en alarmsignal tas emot från annan utrustning. Alarmsignalen kan vara en likspänning mellan 6 och 18 volt.

2. Ifall ingen svarar eller avvisar samtalet eller om samtalet inte når fram så skall alarmet ringa igen oändligt antal gånger med ett par minuters mellanrum så länge användaren inte har kvitterat bort alarmet.

3. För att kvittera ett mottaget alarm så ringer man tillbaka till det uppringande alarmet. Man ska inte behöva svara för att kvittera. Man ska även kunna kvittera bort ett alarm på plats genom att trycka på en knapp.

4. Alarmet drivs med spänning från elnätet och skall automatiskt kopplas till batteridrift vid elavbrott. Förutom kvitteringsknappen ska alarmet ha en av/på-knapp så man kan spara på batteriet vid långvariga elavbrott.

5. Alarmet ska också ha en knapp för testning av alarmets uppringningsfunktion.

6. Alarmet bör ha två indikatorer: en som visar om alarmet är på och en som visar om alarmet är aktivt (okvitterat).

7. Alarmet ska använda sig av GSM-nätet.

2 Teori

I det här kapitlet presenteras och beskrivs teorin som behövs för att kunna förstå de metoder och tillvägagångssätt som utgör lösningen av uppgiften.

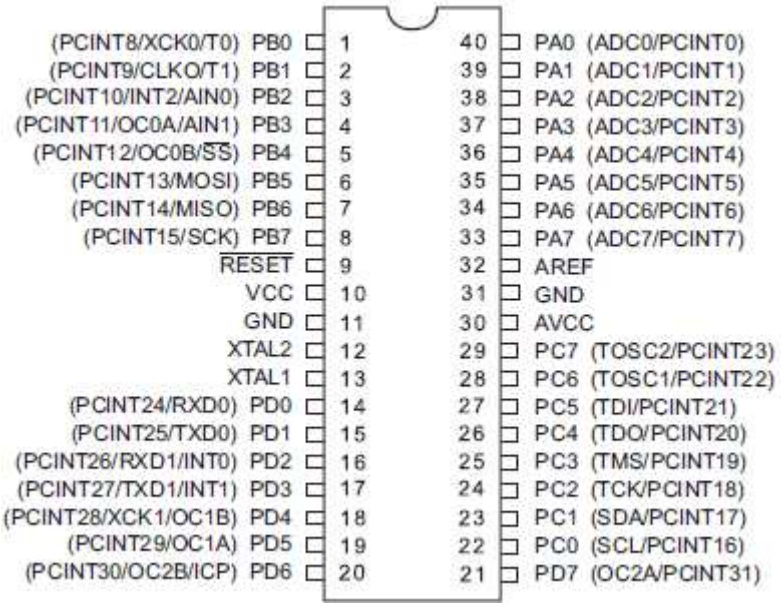
2.1 Huvudkomponenter

2.1.1 ATmega644P

Atmega644P är en 8-bit mikrokontroller producerad av Atmel. Den har 64kB flash programminne, 2kB EEPROM och 4kB SRAM. Några av dess egenskaper är att den har 32 binära I/O-portar, 8-kanals 10-bit AD (Analog till Digital) omvandlare, två programmerbara USART (*Universal Synchronous/Asynchronous Receiver/Transmitter*) portar samt två 8-bit timer och en 16-bit timer. Den har en intern oscillator som ger en klockfrekvens på 8 MHz.

Den kan matas med spänningarna 2,7–5,0 volt och med hjälp av en extern kristalloscillator kan den arbeta med 20 MHz klockfrekvens, men då krävs att matningsspänningen är minst

4,5 volt. Den kan programmeras via JTAG- och SPI gränssnitt. Den finns tillänglig i PDIP-förpackning och några olika ytmonterade förpackningar. Här nedan visas PDIP-förpackningen. /1/



Figur 1. Beskrivning av ATmega644P's pinnar i PDIP-utförande.

Mikrokontrollern har flera funktioner för de flesta pinnar, t.ex. pin #16 hanterar två olika externa avbrott, en I/O och USART1:s mottagare.

Mikrokontrollerns digitala ingångar tolkar en spänning mellan +3 volt och +5,5 volt som en binär etta och en spänning mellan -0,5 volt och +1,5 volt som en binär nolla. Dessa värden gäller när matningsspänningen är 5 volt. /1/

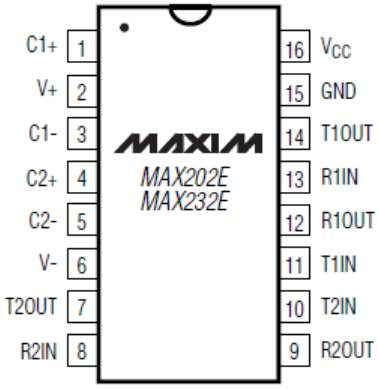
Mikrokontrollern består av ett antal register som alla är uppbyggda av åtta bitar. Bland dessa register finns s.k. *Control Register* (inställningsregister) som styr hur mikrokontrollerns kringutrustning fungerar. Genom att ändra på innehållet i dessa *Control Register*, så kan man t.ex. ändra *baud rate* för en USART, ändra hur snabbt en timer räknar eller ändra när och hur ett externt avbrott inträffar. /1/

En USART är en krets som sköter om seriekommunikation, det finns som sagt två av dessa i en ATmega644P. En USART kan fungera både synkront (synkronisering med klocksignal) eller asynkront (synkroniserar med startbiten). Till en USART:s uppgift hör att skicka och ta emot data seriellt. Mikrokontrollern i övrigt bearbetar alltid data parallellt så en USART omvandlar all seriell data till parallell data vid mottagning och tvärtom

(parallell till seriell omvandling) vid sändning. Den ser också till att de seriella bitarna skickas med rätt hastighet, dvs. med önskad baudrate. Till dess uppgift hör även att tillföra start- och stopbitar och eventuella paritetsbitar till bitströmmen.

2.1.2 MAX202

MAX202 är en spänningsomvandlare. Den omvandlar spänningsnivåerna definierade i RS-232 standarden till TTL-spänningsnivåerna noll eller fem volt. Den omvandlar även andra vägen, dvs. från TTL till RS-232. Denna krets kan användas till att omvandla spänningarna som skickas från en mikrokontrollers USART så att RS-232 standarden följs och möjliggör kommunikation med alla enheter som följer den vanligt förekommande RS-232 standarden.



Figur 2. Beskrivning av MAX202's pinnar i PDIP utförande. /14/

MAX202 har två kanaler. Varje kanal har en mottagare och en sändare, dessa har i sin tur varsin in- och utgång eftersom kommunikationen sker i full-duplex.

Den är tillänglig i DIP16-förpackning och den kräver fyra externa "charge pump" kondensatorer på 0,1 µF för att höja spänningen. En femte 0,1 µF kondensator (by-pass kondensator) kan placeras in för att leda bort störningar. /14/

2.2 Seriekommunikation

Protokollet RS-232 användes tidigare i hemdatorer som länk mellan datorn och kringutrustning som t.ex. modem eller mus. Som kontaktdon används ofta en DB-9 kontakt och då definieras pinnarna enligt tabellen nedan. /8//13/

Tabell 1. RS-232 standardens pinnar i DB9 kontaktdon.

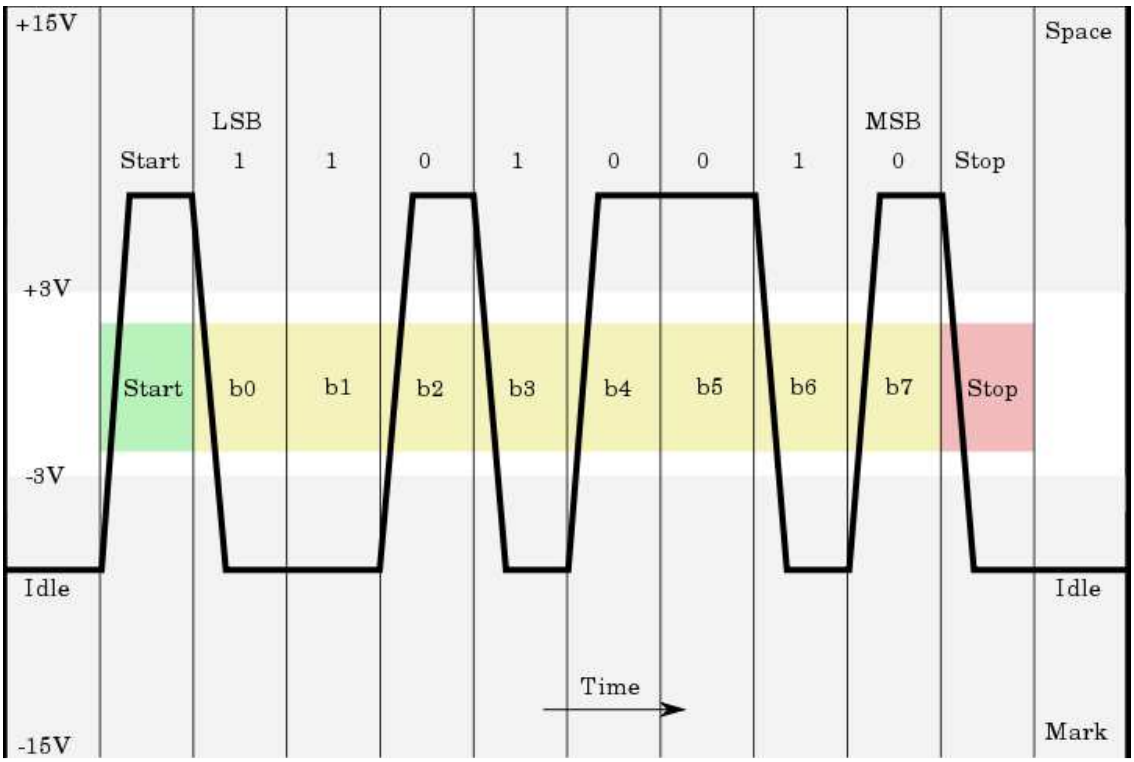
Pin #	Benämning	Namn
1	DCD	Data Carrier Detect
2	RxD	Receive Data
3	TxD	Transmit Data
4	DTR	Data Terminal Ready
5	GND	Signal Ground
6	DSR	Data Set Ready
7	RTS	Request To Send
8	CTS	Clear To Send
9	RI	Ring Indikator

Av dessa nio pinnar behöver man endast tre trådar vid full duplex asynkron kommunikation. En för att skicka data, en för att ta emot data och en jordledning. Pinnarna som behövs är TxD, RxD och GND./8/

RS-232 är ett point-to-point protokoll, vilket betyder att kommunikationen är begränsad till två enheter. Kommunikation är full duplex, vilket betyder att kommunikation i båda riktningarna samtidigt är möjlig.

Som *line coding* används bipolär NRZ (non-return-to-zero). En spänning mellan -3 volt och -25 volt representerar en binär etta medan en spänning mellan +3 volt och +25 volt representerar en binär nolla./6/

Ett typiskt kodord med en startbit, åtta databitar och en stopbit kan då se ut enligt följande bild /6/. På x-axeln är tiden (utan enhet) och på y-axeln är spänningen i volt. LSB anger den minst signifikanta biten och MSB anger den mest signifikanta biten.



Figur 3. Ett typiskt RS-232 kompatibelt tecken. /12/

Varje kodord representerar ett s.k. ASCII-tecken. ASCII-teckentabellen är standardiserad och definierar varje teckens binära, hexadecimala och decimala motsvarighet. Som exempel kan nämnas att kodordet i bilden ovan är 01001011. Slår man upp denna bitsekvens i en ASCII-tabell kan man konstatera att den representerar tecknet K./18/

För att två seriellt kommunicerande enheter ska förstå varandra måste de använda samma specifikation på kommunikationen. Man måste definiera antal start- och stopbitar, paritetsbitar, antalet databitar, flödeskontroll och baud rate. En enkel sådan definition kan vara: en startbit, åtta databitar, en stopbit, ingen paritetsbit, ingen flödeskontroll och hastigheten är 2400 bps (bitar per sekund).

2.3 AT-kommandon

Dennis Hayes utvecklade 1977 ett sätt att både skicka kommandon till det lokala modemet och skicka och ta emot data genom modemet i en och samma datalänk. Eftersom varje modemkommando började med ”AT” (kort för engelskans attention=lystring) blev hans kommandon kända som Hayes AT-kommandon. /5/

Denna uppsättning kommandon eller kommandospråk har regler som måste följas, några av de viktigaste är: /5/

- Alla kommandon börjar med ”**AT**” eller ”**at**”.
- Alla kommandon måste avslutas med ASCII-koden för vagnretur (0x0d).
- Flera kommandon kan kombineras på samma kommandorad.
- En kommandorad får vara maximalt 40 tecken långt.

Och några av de vanligaste kommandon är: /15/

- **ATA**, A=Answer, Svarar inkommande samtal.
- **ATD**, D=Dial, Ringer upp ett telefonnummer t.ex. ATD1234567890.
- **ATH**, H=Hang up, Kopplar ifrån samtalet.

För att skicka dessa kommandon över ett seriellt gränssnitt så skickar man de enskilda tecknens ASCII-kod i en sekvens efter varandra.

3 Lösning

3.1 Lösningsöversikt

Kretsens huvudkomponent är en mikrokontroller. Mikrokontrollern kommunicerar via ett seriellt gränssnitt med GSM-modemet i en mobiltelefon. Mobiltelefonen ringer och tar emot samtalen, men den styrs helt och hållet av mikrokontrollern genom att skicka AT-kommandon. Eftersom det vanligt förekommande RS-232 protokollet används så görs byte av mobiltelefon till en annan modell senare, om det skulle behövas, enklare.

När en extern enhet genererar en alarmsignal så tas den emot av en ingångskrets som anpassar signalen för mikrokontrollerns digitala ingång. En avbrottssubrutin körs omedelbart i mjukvaran i mikrokontrollern när alarmsignalen registreras. Denna avbrottssubrutin skickar AT-kommandot för uppringning till GSM-modemet.

Ett avbrott inträffar i mikrokontrollern även när en kvittering registreras via USART:en och ett annat avbrott inträffar när man manuellt kvitterar alarmet. Vid båda avbrotten ska samma procedur utföras, dvs. alarmet ska kvitteras och nollställas.

Spänningen från elnätet transformeras ner till 12 volt likspänning med en extern transformator. En spänningsregulator ger ut +5 volt till bl.a. mikrokontrollern. Ett 9 volt batteri används som reserv vid spänningsavbrott från elnätet. Dioder separerar batteriet från 12 volt spänningen som kommer från transformatorn.

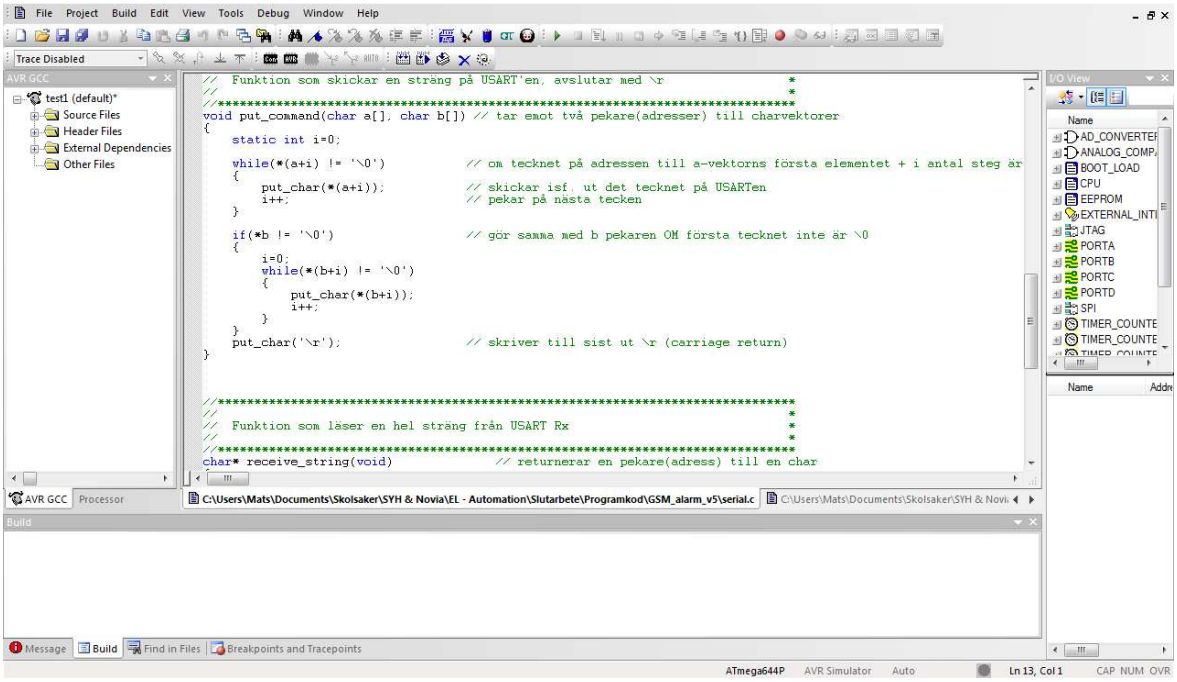
Mobiltelefonens laddare ersätts med en spänningsregulator som förser telefonen med lämplig spänning. Spänningsregulatorn är ställbar för att man enkelt ska kunna byta ut mobiltelefonen till en annan modell.

3.2 Verktyg

3.2.1 AVR Studio 4 / WinAVR

Atmels mjukvara AVR Studio 4 är en IDE (Integrated Development Environment) för Atmels AVR 8 bitars processorer i Windows-miljö. Den har integrerad assembler och en simulator. /2/

Som plug-in kan man installera Win-AVR. Win-AVR är en uppsättning verktyg som inkluderar bl.a. AVR-GCC (en C kompilator), AVR-AS (en assembler) och AVR-GDB (en debugger). C-kompilatorn gör så att man kan programmera mikrokontrollern med C-kod. /17/



Figur 4. AVR Studio 4 med WinAVR.

AVR Studio 4 är enkelt att lära sig och att använda och den har stöd för alla programmerare som stöder 8 bitars AVR-arkitekturen.

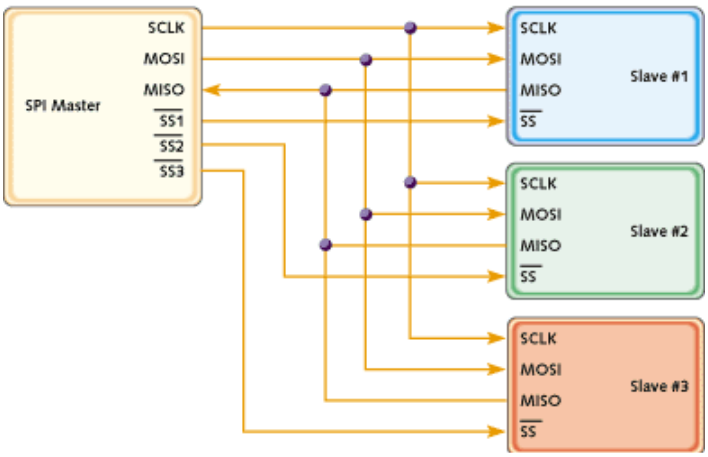
3.2.2 AVRISP mkII

För att överföra programkod skriven i AVR Studio 4 eller annat program till mikrokontrollern så måste det finnas en länk mellan dem. Den länken kan t.ex. vara Atmels AVRISP mkII, en programmerare som överför kod till mikrokontrollern. ISP står för In System Programming och betyder att mikrokontrollern kan programmeras då den är igång.

Figur 5. Atmels AVRISP mkII. /4/

Den kopplas till mjukvaru-utvecklarens PC med en USB-kabel och till processorn kopplas den via SPI (Serial Peripheral Interface) gränssnittet som finns på pinnarna fem till åtta på en Atmega644P.

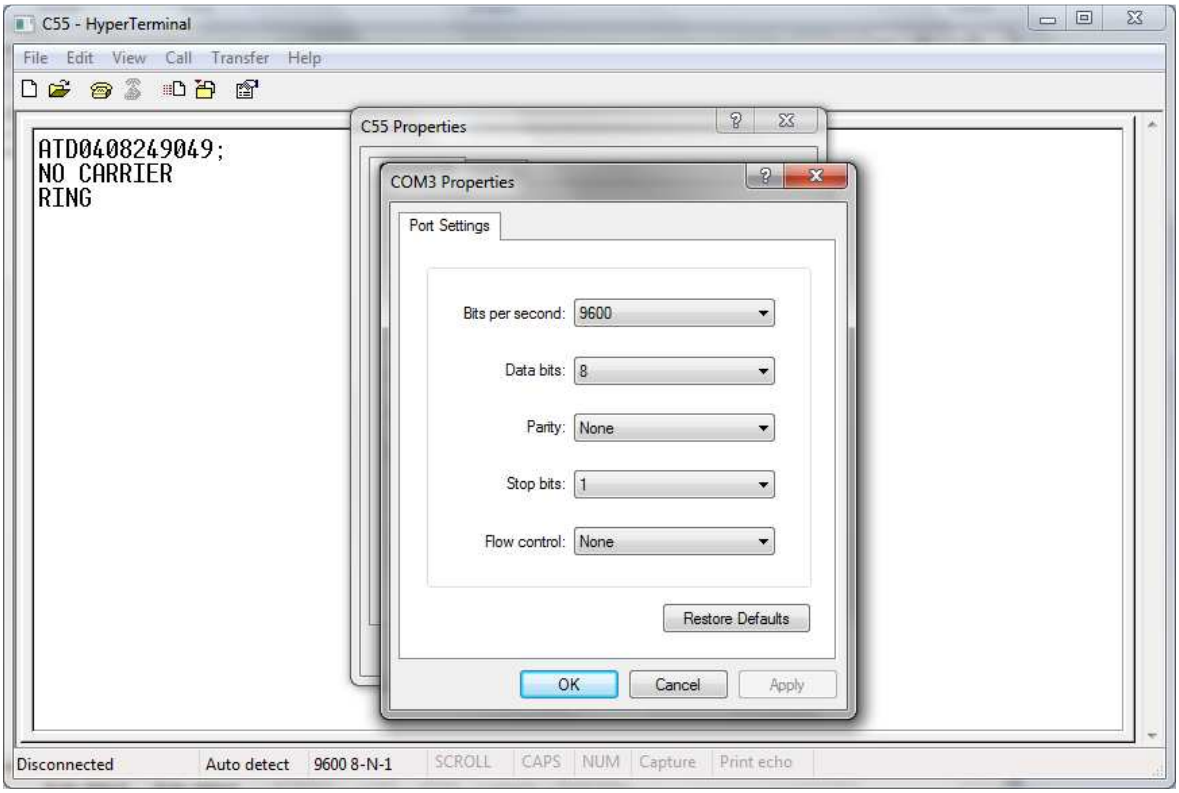
SPI är ett kommunikationsprotokoll som är upprättad av Motorola. Anordningar som följer SPI-standarden kommunicerar enligt master/slave-principen. Standarden definierar fyra pinnar, de är SCLK, MOSI, MISO och SS. SCLK är en klocksignal som behövs eftersom SPI standarden kör med synkron seriekommunikation. MOSI står för Master Out Slave In och transporterar data från Master till Slave. MISO är tvärtom Master In Slave Out och transporterar data från Slave till Master. SS betyder Slave Select och används när det finns flera slavar. /11/



Figur 6. SPI-kommunikationsprotokoll. /11/

3.2.3 HyperTerminal

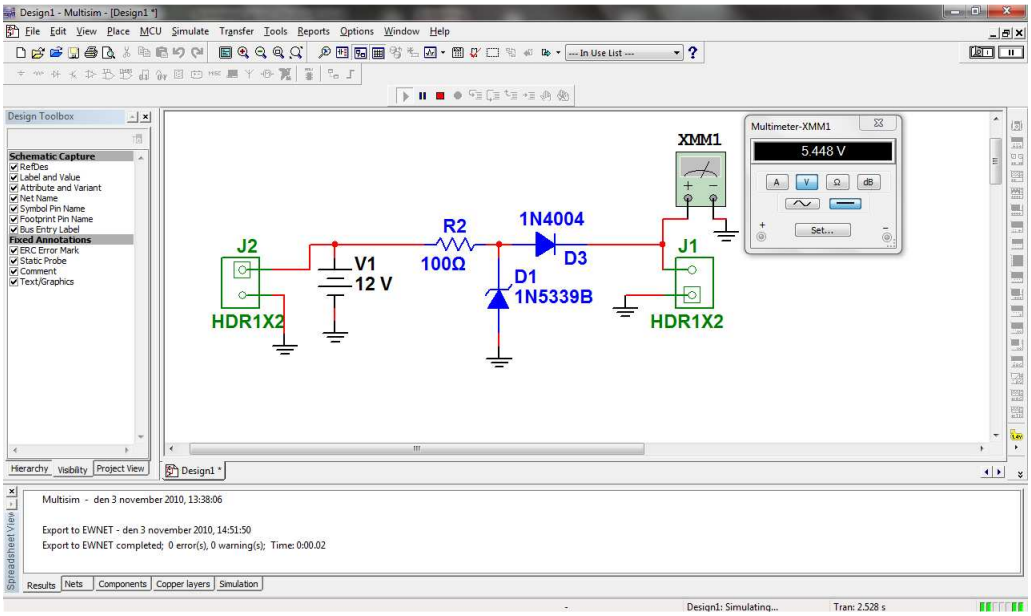
HyperTerminal är ett terminalprogram. Ett terminalprogram används för att skriva till och läsa av serieporten på en PC. HyperTerminal används idag mest för att konfigurera modem och för att testa och felsöka anslutningar. Förr användes HyperTerminal för att koppla ihop datorer, i telnet och BBS (Bulletin Board Systems) sammanhang. /16/



Figur 7. Hyperterminal.

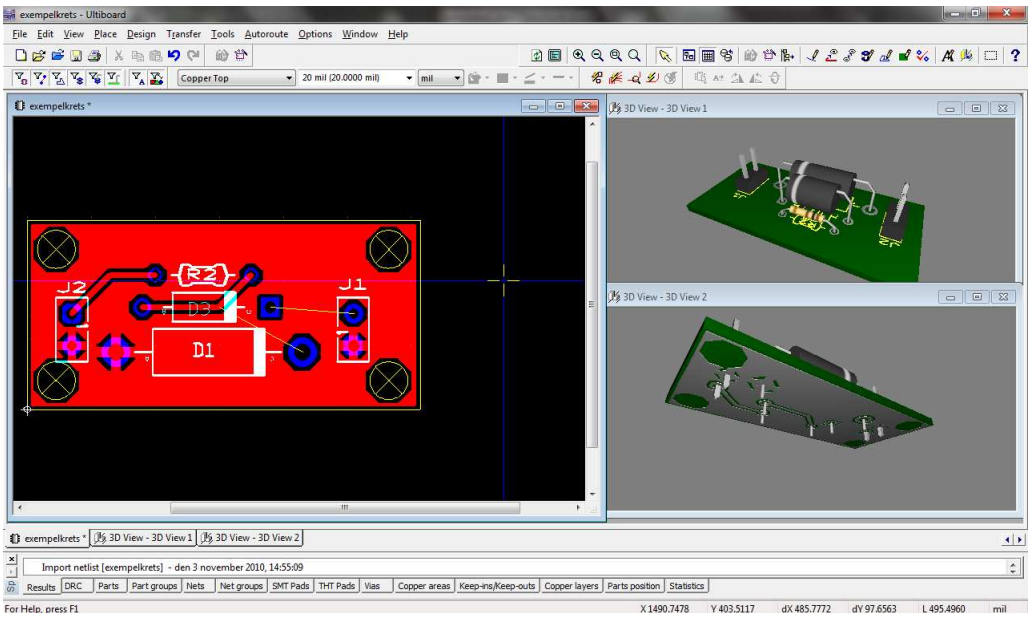
3.2.4 Multisim 11.0 / Ultiboard 11.0

Med National Instruments Multisim ritar man elektriska kretsar. Man placerar in de komponenter man vill ha med i sin krets och kopplar sedan ihop dem med varandra. Man kan placera in olika mätare, t.ex. oscilloskop eller multimeter och simulera kretsen för att se om den fungerar.



Figur 8. National Instruments Multisim 11.0.

När man har kretsens schema ritad i Multisim kan man exportera den till Ultiboard. Ultiboard är ett kretskortsplaneringsprogram med vilket man kan planera och rita kretskort för sin elektriska krets. Man får själv placera komponenterna på kretskortet där man vill ha dem.



Figur 9. National Instruments Ultiboard 11.0.

När komponenterna är utplacerade börjar man rita kopparbanorna mellan komponenternas ben. Programmet hjälper till med att hålla reda på vart alla komponentben ska kopplas. Programmet säger också till om man har gjort något misstag, t.ex. om man har placerat två komponenter för nära varandra eller om kopparbanorna är för nära varandra. Man kan också se kretskortet i 3D. Man kan rotera det virtuella kretskortet fritt för att t.ex. kontrollera utrymme för komponenterna på kretskortet.

3.3 Planering av kretsen

Den elektriska kretsen som utgör grunden för det uppringande alarmet ritades i Multisim. Vissa delar av kretsen är också simulerade i samma program, t.ex. alarmsignalens ingång och spänningsförsörjningskretsarna. Andra delar är testade i laboratoriemiljö och spänningar och signaler är uppmätta med oscilloskop och multimeter.

I detta kapitel kommer specifika delar av kretsen att visas för att beskriva just de delarna. I bilaga B kan hela kretsschemat ses som helhet.

3.3.1 Val av mikrokontroller

Valet av mikrokontrollern beror naturligtvis på vilka egenskaper och funktioner som krävs av den. Minimikraven för mikrokontrollern är följande:

- en USART
- två externa avbrott
- minst en 8-bit timer
- stöder 5 volt matningsspänning
- två digitala utgångar
- *Through-hole* förpackning.

Eftersom jag har tidigare erfarenheter med Atmels Mega AVR-serie mikrokontroller så har jag riktat in mig på dessa. Jag har ett par att välja mellan, av vilka alla har liknande egenskaper. Det som skiljer dem åt mest är flash-minnets storlek. Det bestämmer hur stort program man kan programmera in i den.

AVR Studio 4 räknar ut hur stort programmet är när man kompilerar det. Programmet som är skrivet åt det uppringande alarmet är mindre än 2 kB och kräver under 50 byte RAM-minne. Med dessa siffror kan man konstatera att alla AVR ATmega mikrokontrollrar duger. Jag beslöt mig för att använda en ATmega644P. /3/

3.3.2 Val av klockfrekvens

Eftersom mikrokontrollern har en intern oscillator så behöver man ingen extern, man slipper därmed tre komponenter på kretskortet, oscillatorn och två kondensatorer. Den interna oscillatorn är 8 MHz, men kan också skalas ner till 1 MHz genom att sätta CKDIV8 *fuse bit* till 1, vilket dividerar systemklockfrekvensen med 8.

En *fuse bit* är en inställningsbit genom vilka man kan ställa in olika funktioner i en mikrokontroller. ATmega644P har 19 *fuse* bitar. Några exempel på *fuse bits* är CKSEL0-3 (väljer klockkälla) och JTAGEN (möjliggör JTAG-gränssnittet).

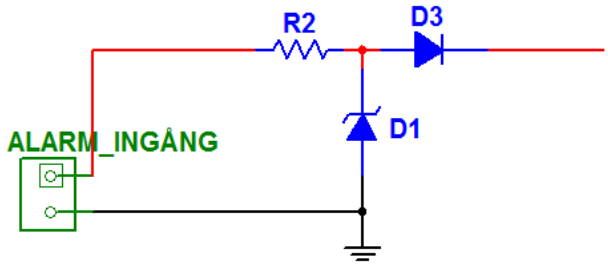
Man måste identifiera vilka funktioner som är beroende av en viss klockfrekvens för att fungera korrekt. Det enda som i det här fallet är tidskritiskt är seriekommunikationen. Det kan nämnas att med en 1 MHz klockfrekvens kan man kommunicera med en hastighet på upp till 576 000 bps, vilket är mycket mera än 9600 bps som krävs här. /1/

Valet står alltså mellan klockfrekvenserna 8 MHz och 1 MHz, men valet beror inte på baud raten eftersom baud rate-generatorn tar klockpulserna direkt från systemoscillatorn oberoende av CLK/8 *fuse bit*. Man kan därför köra processorn med 8 MHz eller 1 MHz klockfrekvens men baud rate-generatorn arbetar ändå alltid med 8 MHz.

Det finns en viss frekvens på oscillatorer med vilken man uppnår 0 % felprocent i seriekommunikationen. Man behöver då ha en extern oscillator. Mer om denna frekvens i kapitel 3.5.2 (USART).

3.3.3 Alarmsignal ingång

Enligt funktionsspecifikationen kan alarmsignalen vara en likspänning på 6–18 volt. För att anpassa denna signal till mikrokontrollerns digitala ingång behövs en spänningsreglerande krets. När alarmsignalen är låg ska den digitala ingången till mikrokontrollern vara en binär nolla. Då alarmsignalen är hög (6–18 volt) ska ingången på mikrokontrollern läsas som en binär etta.



Figur 10. Alarmsignal ingång.

En 5,6 volt zenerdiod D1 håller en konstant spänning på 5,6 volt över sig. Ett spänningsfall på 0,7 volt över dioden D3 sänker utspänningen till 4,9 volt, vilket är en binär etta sett från mikrokontrollern.

Motståndet R2 reglerar strömmen genom zenerdioden. Minsta värdet på R2 beräknas så att strömmen genom D1 inte övergår den i databladet maximalt tillåtna strömmen 865 mA då inspänningen är den maximala dvs. 18 volt. /20/

$$R_{2,min} = \frac{18V}{0{,}865A} \approx 20{,}8\Omega$$

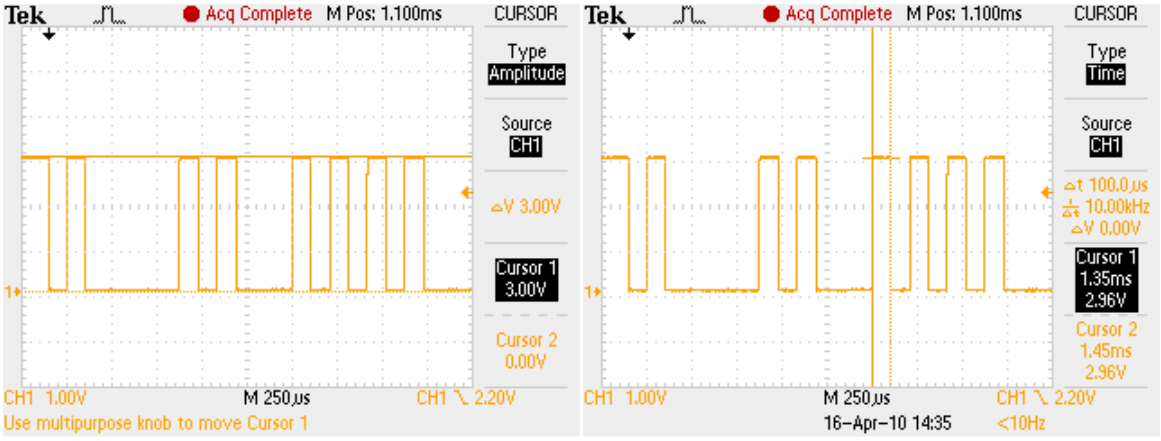
När inspänning är noll är strömmen genom zenerdioden också noll, vilket betyder att denna spänningsreglerande krets ger en utspänning på 0 volt (binär nolla).

3.3.4 Kommunikation

Som GSM-modem används det modem som finns i en Siemens C55 mobiltelefon, den har emellertid ett seriekommunikationsgränssnitt som inte följer RS-232 standarden.

Figur 11. Siemens C55 mobiltelefon. /7/

Genom att med oscilloskop mäta spänning och bitlängd i tid för mobiltelefonens seriekommunikation erhölls specifikationerna för den. Spänningsnivåerna mättes först och resultatet visar att det är en spänning som antingen är noll volt eller +3 volt. För att få reda på vilken baudrate som används av telefonen mättes bredden på en bit. Det visade sig vara i närheten av 100 µs, $\frac{1}{100\mu s} = 10000\ bps$. Genom att med dator framgångsrikt kommunicera med telefonen med den vanligt förekommande hastigheten 9600 bps, konstaterades att telefonen använder sig av just denna *baud rate*.



Figur 12. Oscilloskopbilder av Siemens C55 seriekommunikation.

Jag kunde i detta skede fastslå parametrarna för seriekommunikationen i mobiltelefonens GSM-modem (se tabell 2).

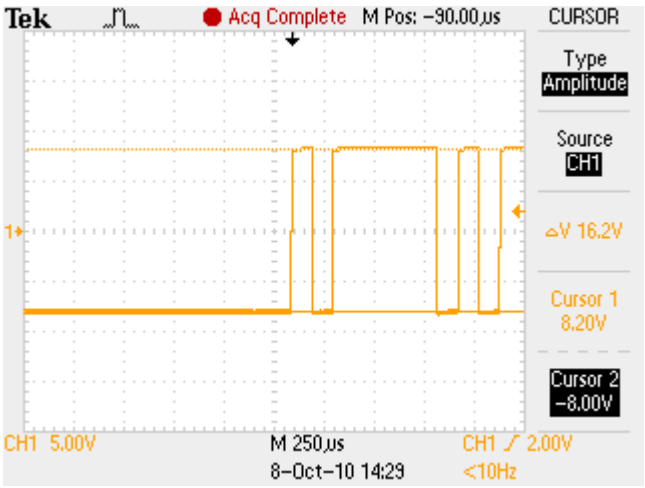
Tabell 2. Parametrar för GSM-modemets seriekommunikation.

Parameter	*värde*
Startbitar	Ja, 1 st.
Stopbitar	Ja, 1 st.
Paritetsbitar	nej
Databitar	8 st.
Baudrate	9600 bps

För att kunna styra och kommunicera med GSM-modemet utan att använda telefonens knappsats, kan man koppla ihop den med en mikrokontroller via en datakabel som har inbyggd krets för RS-232 kompatibilitet eller så kan man koppla telefonen till mikrokontrollens USART direkt, eller nästan direkt, eftersom spänningsnivåerna överensstämmer relativt bra. Mobiltelefonens spänningsnivåer är 0 volt och +3 volt och USART:ens spänningsnivåer är 0 volt och +5 volt. Trots möjligheten till detta enklare förfarande användes ändå RS-232 alternativet för att kunna stöda flera mobiltelefoner (De flesta äldre mobiltelefoner har datakablar för RS-232 kompatibilitet). Siemens egna datakabel DCA-500 användes. Den har en DB9-honkontakt i ena änden och en Siemens kontakt som passar i mobiltelefonen i den andra.

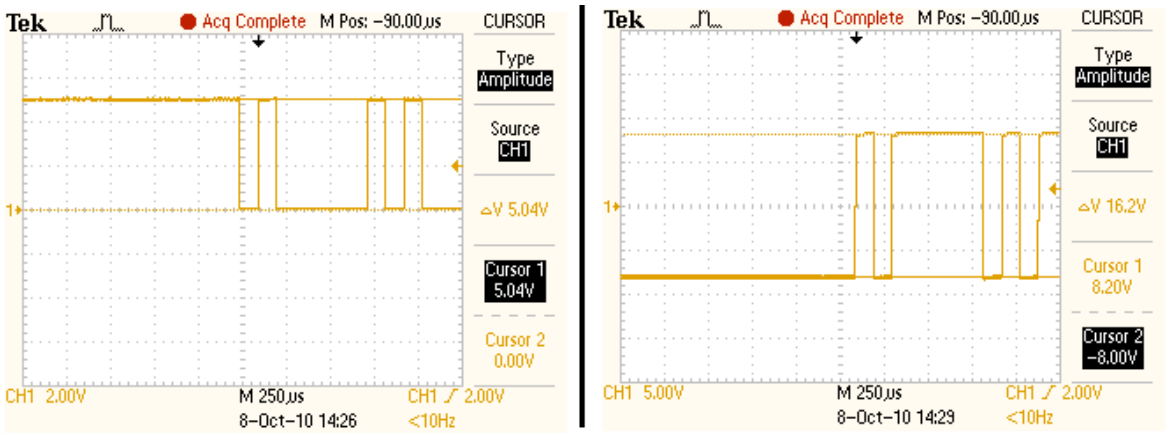
Figur 13. Siemens DCA-500 datakabel. /9/

Spänningarna som datakabeln ger ut i RS-232 änden är ungefär +8 volt och -8 volt enligt uppmätningar med oscilloskop, se figur 14 nedan.



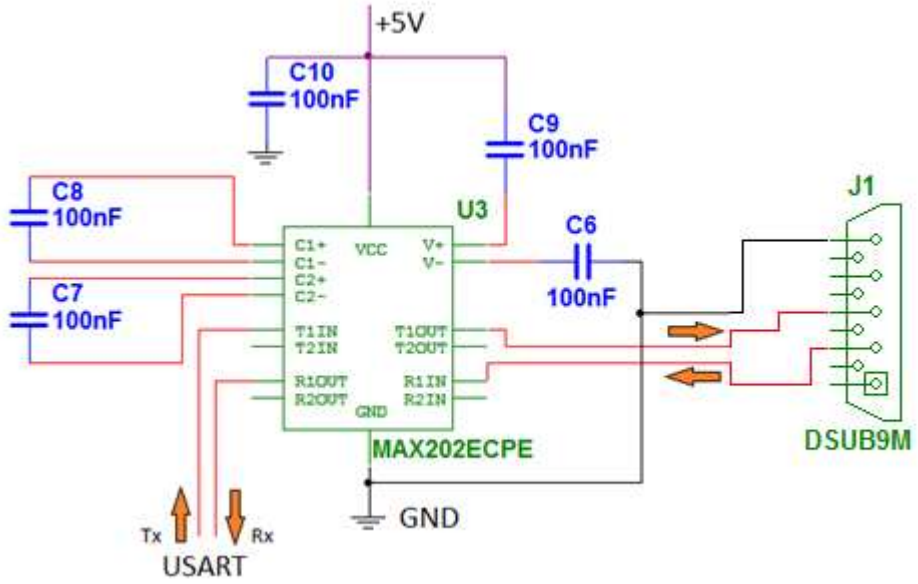
Figur 14. Oscilloskopbild av datakabelns spänningsnivåer i RS-232 änden.

På uppringningsalarmet ska det alltså finnas en DB9-hankontakt där man kopplar in datakabeln. Ur mikrokontrollerns USART kommer TTL-spänningsnivåer (+5 volt, GND). En RS-232 spänningsomvandlare används för att omvandla TTL-nivå till RS-232 kompatibel nivå (±8 volt). /13/



Figur 15. Oscilloskopbilder av spänningsomvandlingen (TTL-RS-232) med MAX202.

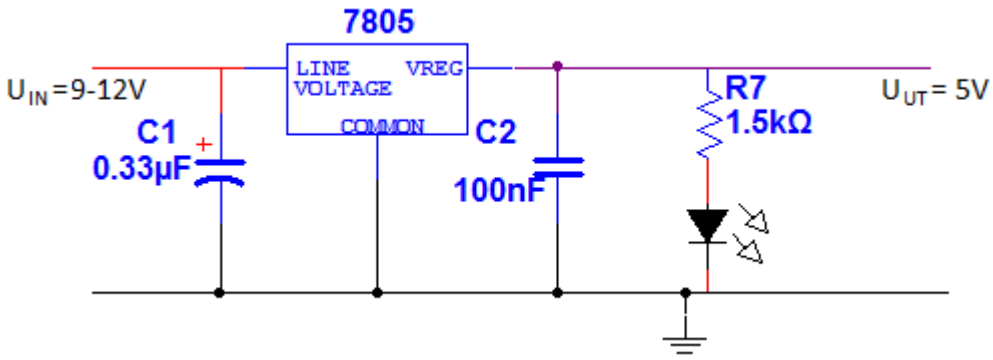
Oscilloskopbilden till vänster visar tecknet ’A’ som kommer direkt ur mikrokontrollerns USART. Signalen översätts med MAX202-kretsen till RS-232 kompatibel nivå och resultatet visas i den högra oscilloskopbilden, notera nollnivån för spänningen i båda bilderna. Kommunikationen andra vägen, dvs. från mobiltelefonen till mikrokontrollern, översätts med samma MAX202-krets till TTL-nivå.



Figur 16. Schema över RS-232 spänningsomvandlarens inkoppling.

3.3.5 Spänningsförsörjning

Spänning ska i första hand tas från elnätet. Den blir nertransformerad till 12 volt och likriktad innan den når det uppringande alarmet. Detta görs alltså med en extern transformator och kommer inte att behandlas här. In i alarmenheten händer emellertid följande.



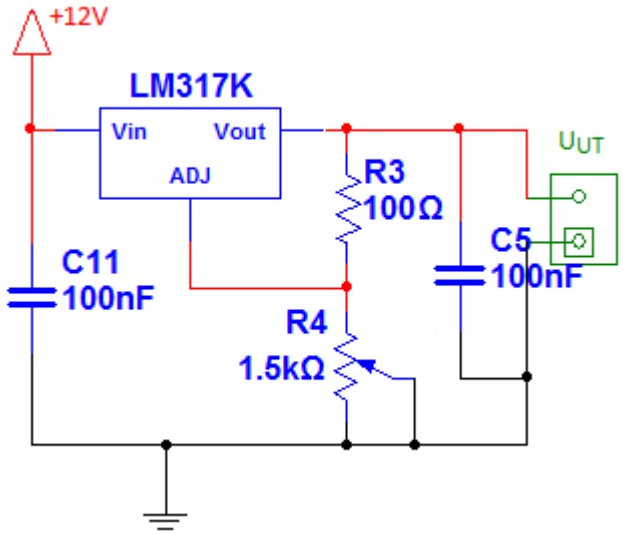
Figur 17. Schema över spänningsförsörjningen till kretsen.

En spänningsregulator (7805) tar ner spänningen från +12 volt till +5 volt. Värde på kondensatorerna C1 och C2 är tagna ur databladet för spänningsregulatorn. Överskottsspänningen förbränns och blir till värme. Spänningsregulatorn kan förbränna 2,0 watt utan kylfläns enligt databladet för komponenten. Man räknar ut hur mycket spänningsregulatorn förbränner enligt följande uträkning. Strömmen som går genom spänningsregulatorn vid normalt alarmtillstånd uppmättes till 20 mA.

$P = \Delta UI = (12 - 5)V * 20mA = 140mW = 0{,}14\ watt$

Eftersom spänningsregulatorn i detta fall förbränner endast 0,14 watt enligt beräkningarna ovan så behövs ingen kylfläns.

En annan spänningsregulator (317T) laddar mobiltelefonens batteri vid behov. Denna regulator har en potentiometer kopplad till sig, med vilken man kan justera utspänningen för att kunna anpassa spänningen för ett flertal olika telefonmodeller. Schemat för den ser ut enligt följande.



Figur 18. Schema över spänningsförsörjningen till mobiltelefonen.

Potentiometern och 100 ohms motståndet är kopplade som en spänningsdelare till regulatorn. Kondensatorerna är avstörningskondensatorer och har inget med regulatorns egentliga funktion att göra. Utspänningen beräknas enligt $U_{UT} = 1{,}25\left(1 + \frac{R4}{R3}\right)$. Med denna konfiguration kan man ställa in spänningar mellan $1{,}25\left(1 + \frac{0}{330}\right) = \mathbf{1{,}25V}$ och $1{,}25\left(1 + \frac{1500}{330}\right) = \mathbf{6{,}93V}$.

När telefonen laddas går en ström på max 460 mA genom spänningsregulatorn 317T. När telefonens batteri är fullt laddat drar den inte lika mycket ström. Man måste räkna ut hur mycket effekt den förbränner i värsta tänkbara scenario. Vid max laddningsström och 12 volt inspänning och 5 volt utspänning förbränner den:

$P = \Delta UI = (12\ V - 5\ V) * 0{,}46\ A = 3{,}22\ W$

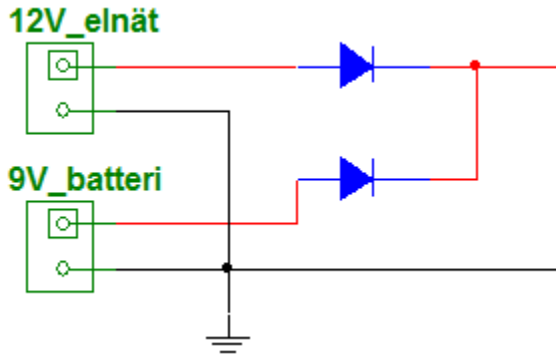
Man behöver då en kylfläns med maximala värmeavledningsförmågan $R_{Th,max}$ på $11°C/W$ enligt följande beräkning.

$$R_{Th,max} = \frac{T_{Junction,max} - T_{omgivning}}{P} = \frac{60°C - 25°C}{3{,}22W} = 11°C/W$$

$T_{Junction,max}$ är maximala temperaturen i spänningsregulatorns kärna, $T_{omgivning}$ är temperaturen för omgivningen och P är effekten som förbränns. Ju mindre värde ($°C/W$) desto bättre värmeavledningsfömåga. /19/

Det faktum att man kan justera utspänningen förenklar ett eventuellt utbyte av mobiltelefonen eftersom en annan mobiltelefonmodell kanske kräver en annan spänning.

Vid ett elavbrott i elnätet ska ett 9 volt batteri ta över omedelbart utan att avbryta någon funktionalitet. En diod spärrar ström från att gå från batteriet eftersom den högre spänningen (+12 volt) blockerar. När elnätets spänning blir lägre än batteriets spänning så tar batteriet över matningen av kretsen.



Figur 19. Schema över backup batteriets inkoppling.

3.4 Planering av kretskortet

Kretskortet ritades i Ultiboard. Det är enkelsidigt och har måtten 86 gånger 71 mm. Storleken på kopparbanorna är 20 mil (ca 0,5 mm) för att tillverkningen av själva kretskortet skulle bli möjlig med den utrustning som fanns tillänglig.

Först delades komponenterna in i byggblock. I ett byggblock ingår komponenter som har att göra med en viss funktion, t.ex. alla komponenter som har att göra med seriekommunikationen är i ett byggblock.

Sedan flyttades byggblocken ihop för att bilda en helhet. Efter viss justering av komponenternas placering började kopparbanorna ritas ut.

Riktlinjer som följdes var följande: /10/

- ➔ göra byggblock av komponenterna
- ➔ placera in byggblocken logiskt
- ➔ jordplan
- ➔ inga 90 graders vinklar på banorna
- ➔ minimera antalet hopptrådar.

3.5 Planering av programkod

Under utvecklingen av programmet för mikrokontrollern användes Hyperterminal på en PC för att emulera ett GSM-modem. Denna metod har den fördelen att man kan se vilka tecken och kommandon som mikrokontrollern skickade vid olika tillfällen, det blev lättare att felsöka programkoden. Hyperterminal användes också för att få reda på vilket kommando eller vilken textsträng GSM-modemet skickar vid inkommande samtal. Det visade sig vara textsträngen ”RING”.

Programmet för mikrokontrollern skrevs helt och hållet med C-kod. Atmels egna AVR Studio 4.0 användes tillsammans med C-kompilatorn Win-AVR. Programkoden strukturerades upp i fem C-filer och fyra h-inkluderingsfiler för att koden skulle bli lättöversiktlig. Filerna är:

- alarm.c
- AT.c
- serial.c
- main.c
- avbrott.c
- alarm.h
- AT.h
- serial.h
- userdata.h

När programmet startar så ställs alla register in. Först ställs portriktningarna in, sedan avbrottsregistren, timern och till sist initieras serieporten USART0. Efter att dessa inställningar har gjorts övergår programmet till en evig slinga. Endast ett avbrott kan få processorn att göra något annat. Avbrott kan inträffa då en alarmsignal kommer, då data skickas till USART0 (från GSM-modemet), då timern har räknat färdigt eller vid manuell kvittering.

Härefter följer en kort förklaring över vad dessa olika avbrott innebär och vad deras avbrottssubrutin gör.

När en alarmsignal utlöser alarmet så försätts alarmet i utlöst tillstånd om alarmet inte redan är i det tillståndet, ägarens telefonnummer rings upp för första gången och timern startar. Vad timern gör kommer vi till strax.

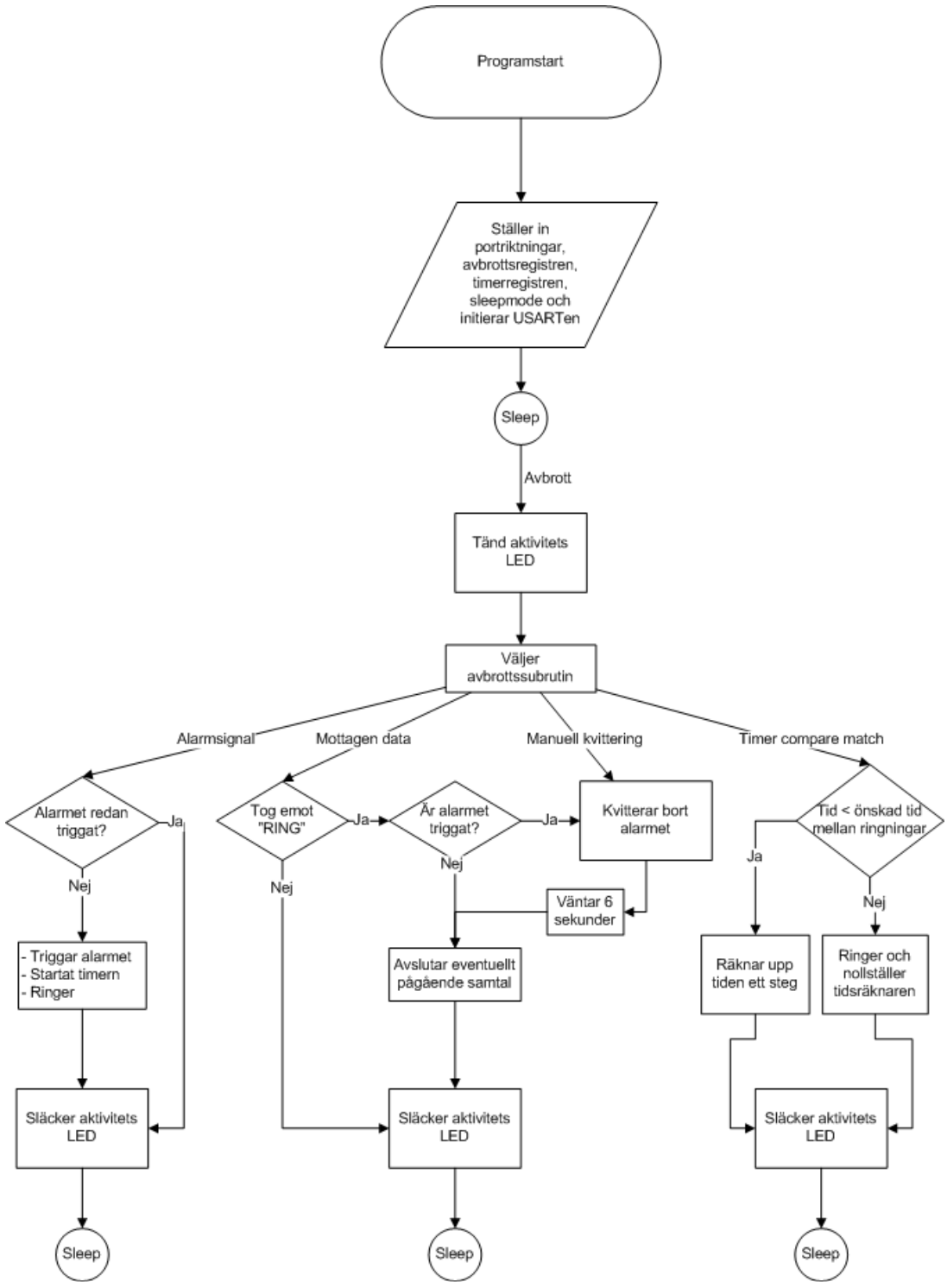
Då data skickas till USART:en så kontrolleras först om mottagen sträng är ”RING”. Om så är fallet så kontrolleras om alarmet är utlöst. Om alarmet är utlöst så kvitteras alarmet bort och efter några sekunder skickas AT-kommandot ”ATH” till GSM-modemet. Om alarmet inte är utlöst så skickas ”ATH” omedelbart. Om något annat än ”RING” tas emot så görs inget. Som parentes kan nämnas att ”RING” är det kommando som GSM-modemet skickar till USART:en när ett inkommande samtal påträffas. Detta används alltså för att kvittera bort ett alarm genom att ringa till GSM-modemet från en telefon.

Timern startas då alarmet utlöses. När timern har räknat upp från noll till sitt fördefinierade värde så skickas ATD, dvs en uppringning utförs, samtidigt nollställs timern så den börjar räkna från noll igen. Vilket det fördefinierade värdet är behandlas i kapitel 3.5.4 (Timer).

Vid manuell kvittering körs kvitteringsproceduren ifall alarmet är utlöst. Timern blir då avstängd och nollställd.

Serial.c filen skrevs för att möjliggöra och förenkla kommunikationen med GSM-modemet. Funktionerna put_char och receive_char sänder respektive tar emot ett tecken på USART0. Med hjälp av dessa funktioner kunde enkelt motsvarande funktioner för hela strängar konstrueras. Funktionen receive_string bygger upp en sträng genom att ta emot tecken tills tecknet för vagnretur påträffas (’\r’). Detta tecken skickas alltid som sista tecken av GSM-modemet. Funktionen put_command skickar en sträng eller ett AT-kommando på USART0.

I userdata.h-filen finns definierat telefonnummer som ska ringas upp vid alarm, antal sekunder mellan uppringningarna och maximalt antal uppringningar som skall utföras per alarm. Detta gör det enkelt att ändra någon parameter, eftersom alla parametrar som man kan vilja ändra på är på samma ställe.



Figur 20. Flödesdiagram för mjukvaran i mikrokontrollern.

3.5.1 Funktioner

En förteckning över funktionerna som används i programkoden och en inledande kort beskrivning över vad de gör.

put_char	Skickar ut ett tecken på USART0's Tx pinne.

receive_char	Tar emot ett tecken från USART0:s Rx pinne.
put_string	Skickar ut en teckensträng på USART0:s Tx pinne.
receive_string	Tar emot en hel teckensträng på USART0:s Rx pinne.
init_USART	Initierar USART:en, dvs. ställer in registren för den så att bl.a. baudrate passar för kommunikation med GSM-modemet.
ring	Skickar, med hjälp av put_string, AT-kommandot för att ringa upp telefonnumret till ägaren.
kvittering	En kvitteringsprocedur som nollställer alarmet.
hangup	Skickar AT-kommandot ”ATH” till GSM-modemet. Detta avbryter pågående samtal.

3.5.2 USART

ATmega644P har två USART-kretsar, USART0 och USART1. Båda kan användas för att kommunicera med GSM-modemet. USART1:s Rx ledning och INT0 har samma fysiska pinne på PDIP-kapseln. Likaså har USART1:s Tx ledning och INT1 samma fysiska pinne. USART0 användes för att INT0 används för manuell kvittering och INT1 reserverades för framtida bruk.

För att ställa in USART0 att följa GSM-modemets seriekommunikationsparametrar behöver man ställa in fyra register. Vilka register man ska ställa in och hur de skall ställas in finns i databladet för mikrokontrollern. Alla register är åtta bitar stora.

Eftersom klockfrekvensen till USART:en genereras genom att dividera systemklockfrekvensen med ett heltal, kan man inte använda exakt 9600 bps med 8 MHz systemklockfrekvens. Förklaringen är att i register UBRR (USART Baud Rate Register) lagras det tal som representerar önskad baud rate enligt följande formel. /1/

$$UBRR = \frac{F_{OSC}}{16BAUD} - 1 \qquad [1]$$

F_{OSC}är systemklockfrekvensen och BAUD är önskad baud rate. Sätter man in aktuella värden i formeln fås $\frac{8MHz}{16*9600} - 1 = 51{,}08333$ ... vilket är ojämnt. Resultatet som lagras i UBRR är avrundat till närmaste heltal, alltså 51. Man kan då räkna ut vilken baud rate man i verkligheten får genom att lösa ut BAUD ur formel [1].

$$BAUD = \frac{F_{OSC}}{16(UBRR+1)} \quad [2]$$

Den *baud rate* man får är:

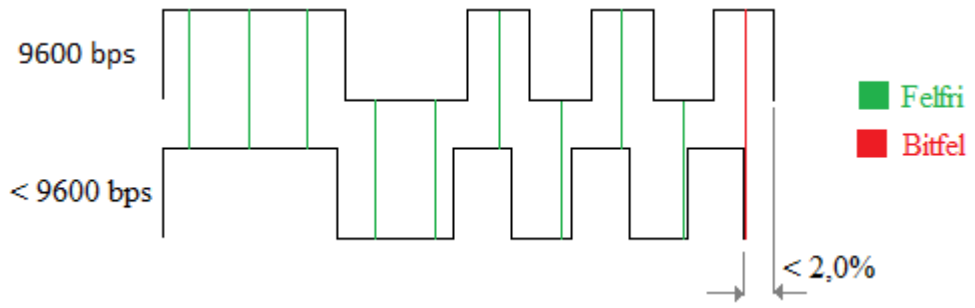
$$\frac{8MHz}{16(51+1)} = 9615{,}38461 \ldots bps$$

Man får alltså ungefär 9615 bps. Enligt ATmega644P:s datablads rekommendation ska felprocenten för mottagaren vara maximalt 2,0 %. Man kan räkna ut vilken felprocent (tidsförskjutningen i bitströmmen) man har med följande formel: /1/

$$ERROR\% = 100\left(\frac{Baudrate_{closestmatch}}{Baudrate}\right) - 1 = 100\frac{9600}{9615{,}38461\ldots} - 1 \approx -0{,}16\%$$

Det betyder att den bitström som erhålls med 9615 bps går 0,16 % snabbare än den med 9600 bps.

För varje tecken som skickas så skickas en startbit, åtta databitar och en stopbit, alltså tio bitar per tecken. Ett tecken med 9600 bps har längden i tid $\frac{1}{9600bps} * 10 \approx 1{,}0417\ ms$ medan ett tecken med 9615 bps har längden $\frac{1}{9615bps} * 10 \approx 1{,}0400\ ms$. Det betyder att när den sista biten har skickats skiljer sig dessa båda bitströmmar $\frac{1}{9600bps} * 10 - \frac{1}{9615bps} * 10$ sekunder, ungefär 1,63 µs. Bilden nedan är inte menad att representera verkligheten utan bara förklara problemet.



Figur 21. Förklarning av problemet med olika baud rates.

Eftersom den aktuella felprocenten är endast 0,16 % (1,63 µs) och det skickas aldrig längre tecken än 10 bitar så är denna konfiguration fungerande. Det kan inte inträffa bitfel vid kommunikationen p.g.a. denna förskjutning, den är för liten.

Om man vill uppnå totalt oförskjuten seriekommunikation kan man använda en extern oscillator med frekvensen 1,8432 MHz eller multiplar av denna frekvens t.ex. 3,6864 MHz

eller 7,3728 MHz. Frånvaron av fel vid dessa frekvenser kommer från det faktum att denna frekvens går jämnt ut i en division med de vanligast förekommande baud raten. Och ett heltal kan exakt representeras av ett 8-bitars register (om det inte är för stort). Låt oss se på ett exempel med en klockfrekvens på 1,8432 MHz.

$$UBRR = \frac{F_{OSC}}{16BAUD} - 1 = \frac{1{,}8432MHz}{16 * 9600} - 1 = 11$$

Om man väljer att UBRR ska vara 11 enligt beräkningen ovan så beräknas den erhållna baud raten ut enligt:

$$BAUD = \frac{F_{OSC}}{16(UBRR + 1)} = \frac{1{,}8432MHz}{16(11 + 1)} = 9600bps$$

Man får alltså exakt 9600 bps. Men eftersom denna frekvens endast finns tillänglig då man använder en extern oscillator så valde jag att köra med den interna 8 MHz oscillatorn, då förskjutningen var så pass låg att kommunikationen fungerade felfritt.

För att aktivera USART0-kretsen och kunna använda den så måste man i programkoden ställa in fyra register:

- UCSR0B USART Control Register 0 B
- UCSR0C USART Control Register 0 C
- UBBR0L USART Baud Rate Register 0 Low
- UBBR0H USART Baud Rate Register 0 High

I registret UCSR0B sätter man bitarna RXEN0 (Reciever Enable) och TXEN0 (Transmitter Enable) till ettor för att aktivera både mottagaren och sändaren för USART0.

I register UCSR0C sätter man, i enlighet med en tabell i databladet, bitarna UCSZ00 (USART0 Character SiZe bit0) och UCSZ01 (USART0 Character SiZe bit1) till ettor för att använda 8-bitars dataord.

I registren UBRR0L (USART0 Baud Rate Register Low Byte) och UBRR0H (USART0 Baud Rate Register High Byte) ställer man in det värde som motsvarar den baud rate man vill använda för USART0. Det här registret diskuterades redan i början av kapitlet. Man hårdkodar inte in värdet i registret utan i stället låter man programmet göra beräkningen. Resultatet lagras i UBRR0L för att sedan högerskifta bitarna åtta steg och lagra dem i UBRR0H.

I registret UCSR0B måste man även sätta biten RXCIE0 till en etta för att tillåta avbrott för mottaget tecken på USART0.

När man nu vill skicka ett tecken på USART'en så lagrar man det helt enkelt i USART0:s dataregister UDR0, så skickas det ut på USART:en. Det är detta som funktionen put_char() gör.

3.5.3 Avbrott

När ett avbrott inträffar avbryts programmets normala gång och en avbrottssubrutin (ISR - Interrupt Sub Routine) körs i stället. Efter att avbrottssubrutinen har exekverats fortsätter programmet där det höll på när avbrottet inträffade. Vad avbrottssubrutinen gör programmeras av programmeraren.

Varje avbrott har ett eget subrutinnamn. I databladet för mikrokontrollern finns alla avbrottssubrutiner uppräknade. Hur man anropar subrutinerna kan ses i bilagan för programkoden (Bilaga 4). Subrutinerna som jag har använt mig av är följande fyra:

Tabell 3. Beskrivning över när avbrottssubrutinerna körs.

ISR	Subrutinen körs:
USART0_RX	då ett helt tecken har tagits emot på USART0
INT2	då spänningen på pin #3 antar högt värde (binär 1:a)
INT0	då spänningen på pin #16 antar högt värde (binär 1:a)
TIMER1_COMPA	då timer 1 har räknat upp till det värde som finns i register OCR1A (Output Compare Register 1 A)

I register EICRA (External Interrupt Control Register A) sätter man bitarna ISCn1 och ISCn0 till ettor om man vill att INTn ska generera ett avbrott på stigande flank, alltså då spänningen stiger.

För att tillåta dessa externa avbrott måste man skriva ettor till bitarna INTn i register EIMSK (External Interrupt Mask Register). Då INT0 och INT2 används skrivs ettor till bitarna INT0 och INT2 i EIMSK-registret.

För att överhuvudtaget få tillåtelse att generera några avbrott måste man skriva en etta till I-biten i SREG-registret. Detta är en global flagga som tillåter avbrott att genereras. I WinAVR finns en funktion som gör detta:

```
sei();        // Set Global Interrupt Enable
```

3.5.4 Timer

Det är meningen att det uppringande alarmet ska kunna ringa upp ägaren flera gånger per alarm med några minuters mellanrum. Till detta behövs en tidsräknare eller timer som räknar tiden mellan uppringningarna.

Timern som har använts finns färdigt i mikrokontrollern, den heter Timer1 och är en 16 bitars timer. Den fungerar parallellt med och oberoende av processorns programexekvering. När alarmet utlöses av en extern alarmsignal så startas timern. Timern stannas och nollställs av programkoden vid kvittering.

Eftersom detta är en 16-bitars timer så kan största värdet den räknar till vara $2^{16} - 1 =$ 65535. Med 7,8125 kHz frekvens på timern skulle det ta $\frac{1s}{7{,}8125\ kHz} * 65535 \approx 8{,}4$ sekunder är räkna till högsta värdet. Varför just 7,8125 kHz används kommer vi till. Men för att det skulle vara enklare att beräkna tiden mellan ringningarna valde jag att ställa in den att räkna till 7813. Det betyder att det blir $\frac{1}{7{,}8125\ kHz} * 7813 \approx 1$ sekund mellan det att avbrotten inträffar (exakt 1 sekund och 64 µs, betraktas härifrån som 1,0 s). Timern genererar då ett avbrott en gång per sekund, men eftersom tiden mellan uppringningarna är flera hundra sekunder så kan man inte utföra en uppringning vid varje avbrott. Ytterligare åtgärder måste alltså vidtas, jag återkommer till dem i slutet av kapitlet.

För att ställa in timern så att den genererar ett avbrott en gång per sekund måste man ställa in följande fyra register.

I TCCR1B-registret (Timer/Counter Control Register 1 B) sätter man biten WGM12 för att använda CTC-mode (Clear Timer on Compare). I CTC-mode räknar timern upp till det värde som finns i register OCR1A, genererar ett avbrott och nollställer timern.

I TCCR1B-registret bestäms också med vilken frekvens timern ska arbeta. Då det handlar om längre tider vill man ha så låg frekvens som möjligt på timern. Lägsta alternativet fås om man sätter bitarna CS12 och CS10, då blir timerns frekvens systemklockfrekvensen delat med 1024. Om systemklockfrekvensen är 8 MHz blir timerns frekvens då 7,8125 kHz. Därför användes 7,8125 kHz i beräkningarna tidigare.

I OCR1A-registret lagrar man det värde som timern skall räkna upp till. I det här fallet talet 7813.

I registret TIMSK1 (Timer Interrupt Mask register 1) sätter man biten OCIE1A (Output Compare Interrupt Enable 1 A) till en etta för att tillåta compare-match-A avbrott för timer 1. Det betyder att när timer 1 och OCR1A har samma värde så genereras ett avbrott.

När ett avbrott väl genereras så hoppar programmet till avbrottssubrutinen för compare-match-A, där en variabel av datatypen int ökar sitt värde med 1. När denna variabel har ökat sitt värde så att det motsvarar det antal sekunder som man vill ha mellan uppringningarna, så utförs en uppringning och både timern och variabeln nollställs och börjar räkna uppåt igen.

3.5.5 Sleep mode

Idén med att använda *sleep mode* är att spara energi, det är alltså ett energisparläge. När mikroprocessorn inte utför några beräkningar kan man i stället för att använda en evighetsslinga låta den stänga ner vissa delar av mikrokontrollern. Man kan stänga ner klockpulserna till t.ex. CPU och ADC.

Det finns några olika energisparlägen som kan användas och de stänger alla av lite olika avdelningar in i mikrokontrollern. För att sedan aktivera mikrokontrollern igen finns det lite olika källor att välja mellan t.ex. externa avbrotten INT0-2 och WDT (Watch Dog Timern).

Det energisparläge som jag använde kallas för *Idle* (tomgång) och är det läge som sparar minst energi av de tillängliga alternativen, men det var nödvändigt att använda *Idle* för det är det enda energisparläge som mikrokontrollern kan vakna upp ifrån av ett avbrott för mottagen data på USART0. Att mikrokontrollern skall kunna ta emot data på USART0 när som helst är ett krav för att alarmet ska fungera.

Idle släcker ner klockpulserna till både CPU och FLASH. För att väcka den används antingen de externa avbrotten INT0 (manuell kvittering) eller INT2 (alarmsignal) eller avbrott för mottagen data på USART0. Den aktiveras också när Timer1 genererar ett avbrott.

4 Resultat och diskussion

Resultatet av detta arbete är en prototyp som uppfyller funktionsspecifikationen till sin helhet. Det finns ännu några saker man kan optimera och ändra på, men som prototyp är det en bra grund att bygga vidare på t.ex. för kommersiellt bruk. Man kan också använda prototypen utan förändringar då den fungerar alldeles förträffligt bra som den är.

Vad man borde tänka på om man fortsätter att utveckla detta uppringande alarm är att man kunde implementera stöd för SMS utskick där man meddelar vad som är felet, då borde man förstås ha flera alarmingångar. Inget är förstås omöjligt. En annan sak som man då kunde göra är att i alarmtillstånd alltid kontrollera varje inkommande samtals telefonnummer och bara kvittera alarmet när det är ägaren som ringer. Det finns AT-kommando som gör detta (AT+CLIP = Calling Line Identification Presentation) som man kunde använda sig av. Tyvärr stöder inte alla GSM-modem detta kommando.

Kretskortet som först planerades blev någorlunda bra. Jag använde mig av 0,25 mm kopparbanor så att jag kunde dra dem mellan komponentben. Inga hopptrådar behövdes. Jag blev tvungen att ändra på kretskortet p.g.a. att utrustningen för kretskortstillverkning inte klarade av så tunna banor. Jag måste använda 0,5 mm kopparbanor, dessutom blev jag instruerad att inte dra banorna mellan komponentben. Jag blev tvungen att göra flera hopptrådar. Jag kan ju också nämna att nuvarande kretskort fungerar även det helt utmärkt.

Ett val som jag blev tvungen att ta ställning till var om jag skulle ha en RS-232 omvandlare och en datakabel åt telefonen eller om jag skulle ha en direktkopplad variant utan datakabel. Jag valde att implementera RS-232 standarden för att ha möjligheten att kunna byta GSM-modemet mot ett annat.

Ett annat val som jag gjorde var vilken systemklockfrekvens som skulle användas. Jag valde att inte ha någon extern oscillator. Detta medförde en liten förskjutning (en felprocent) i seriekommunikationen, men eftersom förskjutningen konstaterades vara för liten för att orsaka bitfel fanns ingen orsak att använda extern oscillator. Under projektets gång har jag aldrig sett att ett enda felaktigt tecken skulle ha tagits emot. Trots detta skulle man ha kunnat implementera felkontroll, men det ansågs vara onödigt.

Ett alternativt sätt att strukturera upp programkoden undersöktes i ett sent skede. Tanken var att rita ett tillståndsdiagram och att utifrån det skapa grunden för programkoden. Idén

slopades för att detta alarm har så enkelt tillståndsdiagram så att ingen grund kunde skapas. Det skulle också ha tagit onödigt lång tid att ändra på den befintliga koden.

Först användes en av de två 8-bits räknarna tillängliga i mikrokontrollern för att räkna tiden mellan uppringningarna. Den 16-bits räknaren togs sedan i bruk fastän den tidigare lösningen fungerade bra. Skillnaden är att nu behöver processorn inte behandla avbrottet för *compare-match-a* 100 gånger per sekund utan bara en gång per sekund. Detta sparar på processortiden. För att ytterligare optimera mjukvaran och utnyttja hårdvaran till fullo kunde en räknare (timer/counter) räkna antal utförda uppringningar i stället för den globala variabel som nu används.

Jag noterade att mobiltelefonen stöder många olika *baud rates*. Det i sig är inget problem, men när den ställer in sig automatisk till synes när som helst blir det ett problem. Det löstes genom att i början av programkoden skicka kommandot AT till GSM-modemet med ”rätt” *baud rate*. Då anpassar sig telefonen till den *baud rate*. Man måste då ha igång GSM-modemet när man startar igång det uppringande alarmet.

Det finns en automatisk funktion för att placera komponenterna på kretskortet i kretskorts-planerings programmet, men personligen så tycker jag att den placerar dem alltför ologiskt för att man ska kunna använda den.

4.1 Hur kraven uppfylldes

Som svar på funktionsspecifikationen i det inledande kapitlet beskrivs här hur de funktionaliteter som krävdes har implementerats och förverkligats. För att inte behöva bläddra tillbaka till kapitel 1 och för att göra kapitlet ännu tydligare är funktionsspecifikationen repeterad här.

1. Alarmet ska kunna ringa upp ett telefonnummer då en alarmsignal tas emot från annan utrustning. Alarmsignalen kan vara en likspänning mellan 6 och 18 volt.

   När en alarmsignal kommer in regleras spänningen ner till ~5 volt av en zenerdiod. Spänningen leds in till ett av mikrokontrollerns ingång. Ett avbrott genereras på stigande flank och subrutinen för mottagen alarmsignal körs. I denna subrutin instrueras ett GSM-modem att utföra en uppringning till det inprogrammerade telefonnumret.

2. Ifall ingen svarar eller avvisar samtalet eller om samtalet inte når fram så skall alarmet ringa igen oändligt antal gånger med ett par minuters mellanrum så länge användaren inte har kvitterat bort alarmet.
   När en alarmsignal har tagits emot så ringer alarmet med ett inprogrammerat antal sekunders mellanrum ett inprogrammerat antal gånger så länge som alarmet inte har kvitterats bort. Mellanrummet kan vara mellan 1 sekund och ca 136 år och antalet uppringningar per alarm kan vara mellan 1 gång och ca 2,15 miljarder gånger och dessutom kan man välja oändligt antal uppringningar.

3. För att kvittera ett mottaget alarm så ringer man tillbaka till det uppringande alarmet. Man ska inte behöva svara för att kvittera. Man ska även kunna kvittera bort ett alarm på plats genom att trycka på en knapp.

   När GSM-modemet tar emot ett samtal så skickar det textsträngen RING till mikroprocessorn. Om mikroprocessorn tar emot textsträngen RING så körs kvitteringsproceduren om alarmet är utlöst. En tryckknapp som är kopplad till en av mikrokontrollers ingångar generar ett avbrott på stigande flank. Avbrottssubrutinen som körs till följd av avbrottet anropar kvitteringsproceduren och alarmet kvitteras och nollställs.

4. Alarmet drivs med spänning från elnätet och skall automatiskt kopplas till batteridrift vid elavbrott. Förutom kvitteringsknappen ska alarmet ha en av/på knapp så man kan spara på batteriet vid långvariga elavbrott.

   Alarmet drivs med en extern 12 volt DC transformator kopplad till elnätet. Parallellt med denna spänning finns ett 9 volt batteri. Dessa två spänningskällor är ihopkopplade via varsin diod. Dioderna förhindrar dels att spänning dras ur batteriet under normal drift och dels 12 volt spänning matas till batteriet. En brytare bryter spänningen efter dioderna. Detta gör att ingen ström dras från varken elnätet eller från batteriet.

5. Alarmet ska också ha en knapp för testning av alarmets uppringningsfunktion.

En tryckknapp är kopplad så att den förser samma ingång som den externa alarmsignalen är kopplad till med spänningen 5 volt. En diod gör så att ström inte går fel väg ut till alarmet som genererar alarmsignalen. När man trycker på knappen simuleras en extern alarmsignal och alarmet utlöses.

6. Alarmet bör ha två indikatorer; en som visar om alarmet är på och en som visar om alarmet är aktivt (okvitterat).

   En lysdiod som lyser när strömbrytaren är på indikerar att alarmet är på och har spänning. En annan lysdiod lyser alltid när alarmet är utlöst. Den senare nämnda slocknar när alarmet kvitteras.

7. Alarmet ska använda sig av GSM-nätet.

   Det uppringande alarmet använder sig av GSM-modemet i en mobiltelefon. Ett SIM-kort med fungerande anslutning till en operatör behövs för att detta skall fungera.

5 Källförteckning

/1/ Atmel. (2010). *8bit AVR Microcontroller with 16/32/64K in-system programmable flash*. http://www.atmel.com/dyn/resources/prod_documents/doc8011.pdf (hämtat: 12.10.2010 kl.15:40).

/2/ Atmel. (u.å.) *AVR Studio 4*. http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2725 (hämtat: 23.10.2010 kl.14:44).

/3/ Atmel Corporation. (2003). *AVR Microcontrollers Selection Guide*. http://www.atmel.com/dyn/resources/prod_documents/doc4004.pdf (hämtat: 24.10.2010 kl.17:07).

/4/ Atmel Corporation. (u.å.). *AVRISP mkII In-System Programmer*. http://www.atmel.com/dyn/resources/prod_images/AVRISPmkII.jpg (hämtat: 23.10.2010 kl.15:42).

/5/ Bies, L. (u.å.). *Hayes modem AT command set*. http://www.lammertbies.nl/comm/info/hayes-at-commands.html (hämtat: 6.10.2010 kl.09:30).

/6/ Björklund, D. (2010). *Datakommunikation*. Vasa: Yrkeshögskolan Novia

/7/ Club-java. (u.å.) Siemens C55. http://www.club-java.com/Public/telephones/images/Siemens/c55.jpg (hämtat: 11.10.2010 kl.16:31).

/8/ Ewert, M. (1998). *Datakommunikation Nu och i framtiden(2)*. Lund: Studentlitteratur

/9/ GermanMobilePhone. (u.å.). *Siemens DCA-500 Serial data cable.* http://www.germanmobilephone.com/images/product_images/popup_images/75_0.jpg (hämtat: 11.10.2010 kl.14:32).

/10/ Jones, DL. (2004). *PCB Design Tutorial* (Revision A). http://www.alternatezone.com/electronics/files/PCBDesignTutorialRevA.pdf (hämtat: 27.10.2010 kl.14:36).

/11/ Kalinsky, D. & Kalinsky, R. (2002). *Introduction to Serial Peripheral Interface*. http://www.eetimes.com/discussion/other/4023908/Introduction-to-Serial-Peripheral-Interface/ (hämtat: 23.10.2010 kl.15:31).

/12/ Ktnbn. (2007). *Rs232 oscilloscope trace.jpg*. http://en.wikipedia.org/wiki/File:Rs232_oscilloscope_trace.svg (hämtat: 03.11.2010 kl.13:45).

/13/ Lindén, H. (2007). *Digitalteknik 2 och mikroprocessorer*(version 3). Vasa: Svenska Yrkeshögskolan

/14/ Maxim. (2000). *±15kV ESD-Protected, +5V RS-232 Transcievers*. http://www.datasheetcatalog.org/datasheet/maxim/MAX200-MAX213.pdf (hämtat: 18.11.2010 kl.11:17).

/15/ Microsoft. (2004). *AT Modem Command Reference*. http://support.microsoft.com/kb/164660 (hämtat: 06.10.2010 kl.09:45).

/16/ Microsoft. (2005). *HyperTerminal overview*. http://technet.microsoft.com/en-us/library/cc736511%28WS.10%29.aspx (hämtat: 11.10.2010 kl.10:35).

/17/ O'Flynn, C. (2004). *Downloading, Installing and Configuring WinAVR*. http://winavr.sourceforge.net/install_config_WinAVR.pdf (hämtat: 23.10.2010 kl.14:53).

/18/ Prata, S. (2005). *C++-programmering, 5:e upplagan(5)*. Sundbyberg: Pagina Förlags AB

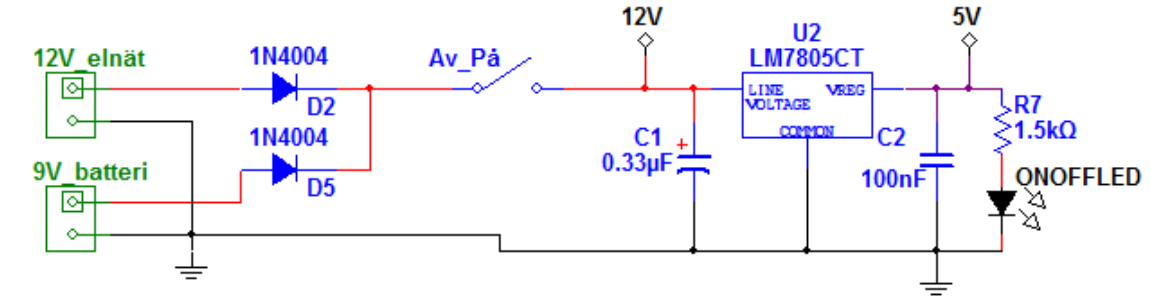
/19/ Renewable Energy UK. (2009). *LM317T Heatsinking Choosing a heatsink for use with an LM317T voltage regulator*. http://www.reuk.co.uk/LM317T-Heatsinking.htm (hämtat: 29.10.2010 kl.14:34).

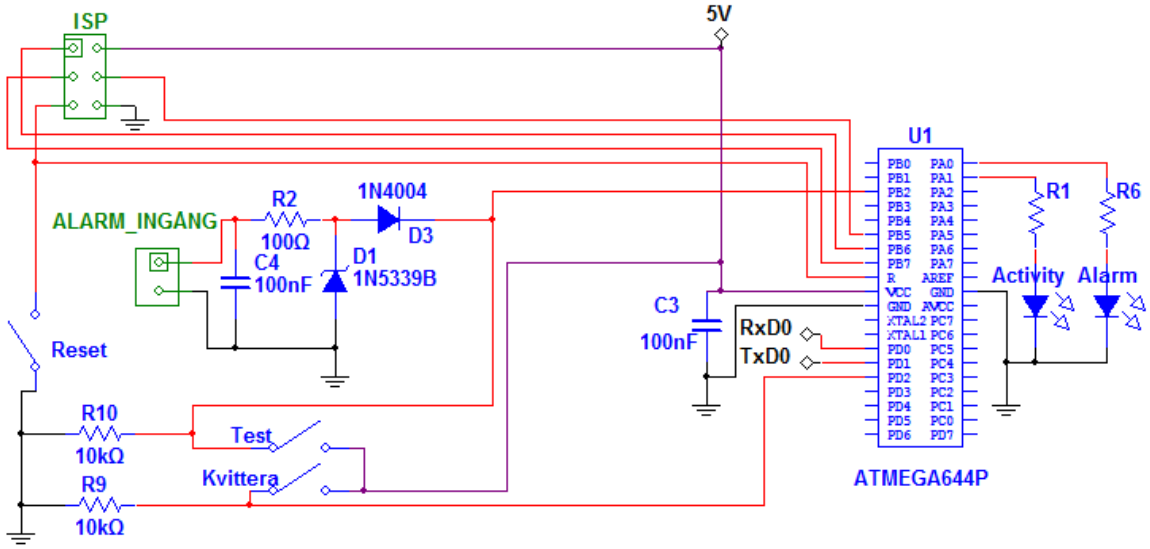
/20/ Semiconductor Components Industries. (2004). *1N5333B Series Preferred Device 5 Watt Surmetic™ 40 Zener Voltage Regulators*. http://www.datasheetcatalog.org/datasheet2/f/0xfq5wloy1rrf34oapwgp3y2lhcy.pdf (hämtat: 18.11.2010 kl.11:31).

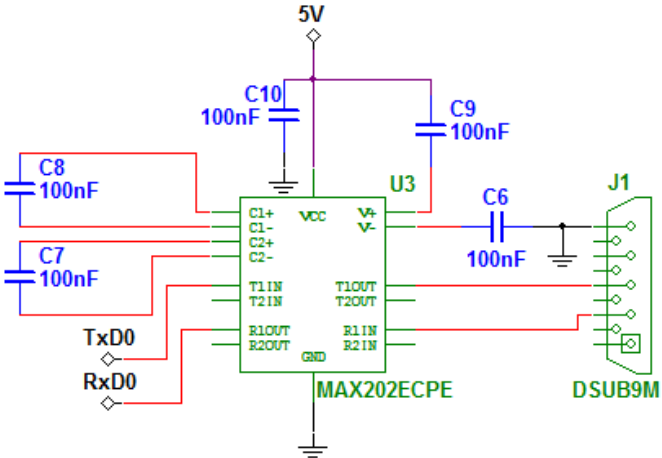
6 Bilagor

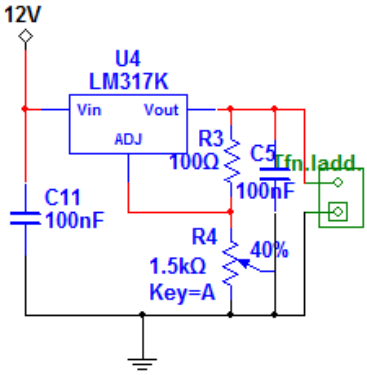
Bilaga A	Komponentförteckning
Bilaga B	Kretsschema
Bilaga C	Kretskortslayout
Bilaga D	Programkod

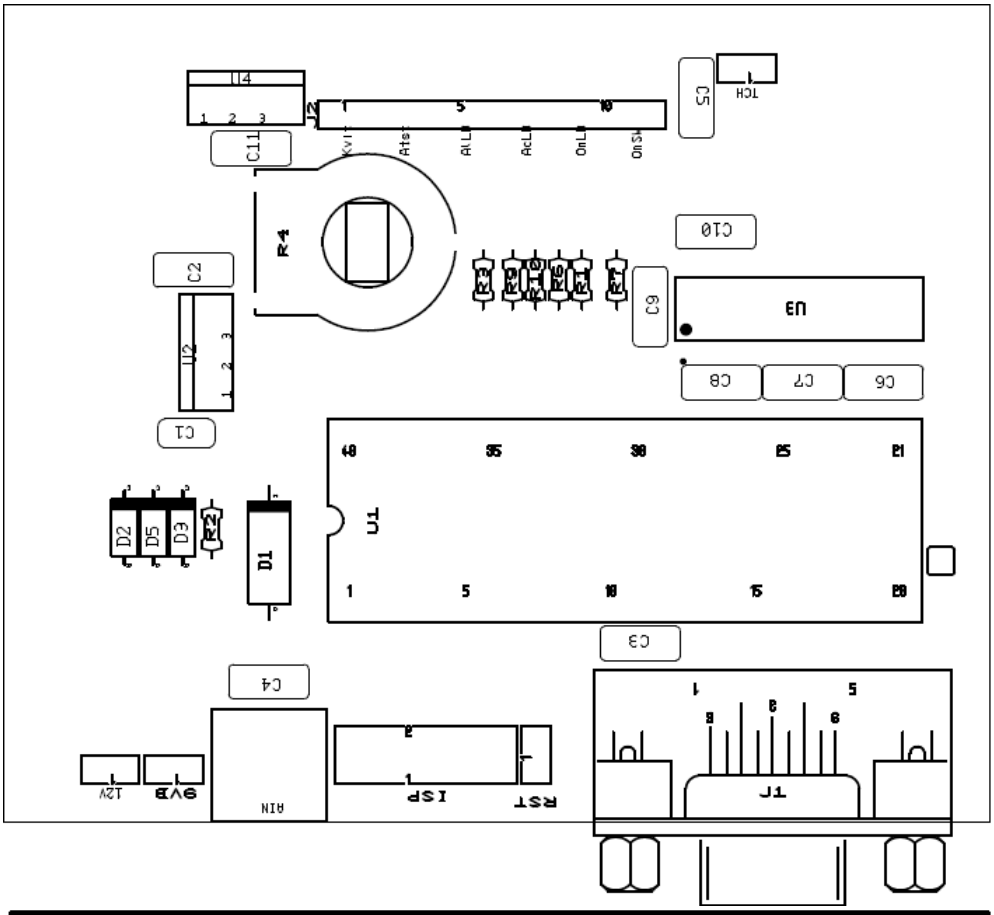
Antal	**Beskrivning**	**Benämning**
1	ATmega644P	U1
1	40 PDIP sockel	
1	Spänningsregulator LM7805CT	U2
1	Spänningsomvandlare MAX202	U3
1	16 PDIP sockel	
1	Spänningsregulator LM317T	U4
3	Motstånd 1,5kΩ	R1, R6, R7
2	Motstånd 100Ω	R2, R3
2	Motstånd 10kΩ	R9, R10
1	Potentiometer 1,5kΩ	R4
1	Elektrolytkondensator 0,33µF	C1
10	Keramisk kondensator 100nF	C2–C11
1	Zenerdiod 1N5339B	D1
3	Diod 1N4004	D2, D3, D5
2	Röd LED	AlLED, OnLED
1	Kontakt HDR2x3	ISP
1	Kontakt HDR1x2	
1	Kontakt DSUB9-Male	J1
2	Tryckknappar (Test, kvittera)	
1	Strömbrytare (Av/På)	
1	Batterihållare 9V	
1	12V DC Kontakt	
1	7°C/W kylfläns TO-220	
1	Kontakt (Terminal Block) 2 pin	
1	GSM-modem (Siemens C55)	
1	Datakabel DCA-500 RS-232	

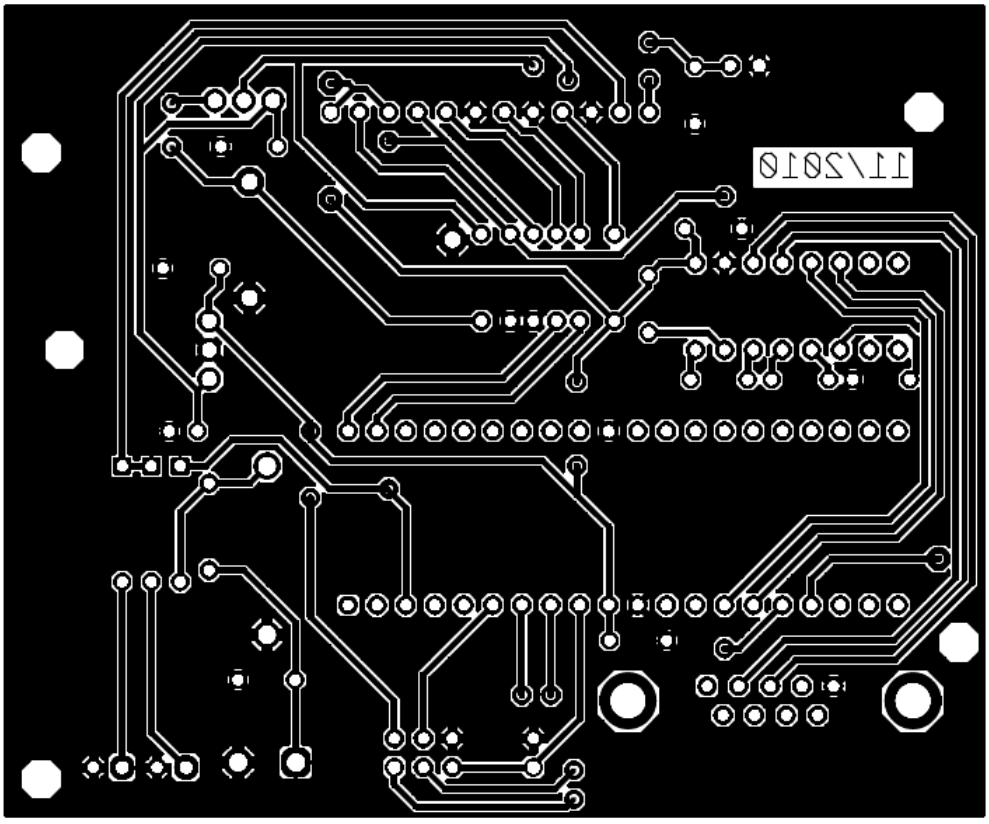

12V_elnät
9V_batteri
1N4004
D2
1N4004
D5
Av_På
12V
C1
0.33µF
U2
LM7805CT
LINE VOLTAGE
VREG
COMMON
C2
100nF
5V
R7
1.5kΩ
ONOFFLED


ISP
5V
U1
ALARM_INGÅNG
R2
100Ω
1N4004
D3
C4
100nF
D1
1N5339B
Reset
C3
100nF
RxD0
TxD0
R1
R6
Activity
Alarm
R10
10kΩ
Test
Kvittera
R9
10kΩ
ATMEGA644P


5V
C10
100nF
C9
100nF
C8
100nF
C7
100nF
U3
C6
100nF
J1
TxD0
RxD0
MAX202ECPE
DSUB9M


12V
U4
LM317K
Vin
Vout
ADJ
R3
100Ω
C5
100nF
Tfn.ladd
C11
100nF
R4
1.5kΩ
40%
Key=A


U4
J2
C11
C5
TCH
Kvi
Ats
ALa
AcLa
OnLa
OnSh
R4
C10
C2
U2
R3
R6
R1
R7
C9
U3
C8
C7
C6
C1
U1
D2
D5
D3
R2
D1
C3
C4
9VB
12V
RIN
ISP
RST
TR


11\2010

alarm.h

```
int ringningsnummer;     // räknar antalet utförda uppringningar
char triggat_alarm;      // global alarmflagga

void kvittering(void);  // funktion som kvitterar och nollställer
```

AT.h

```
void hangup(void);  // funktion som avslutar samtal
void ring(void);    // funktion som ringer upp
```

serial.h

```
// funktion som initierar USARTen
void init_USART(int baudrate, int databits, int stopbits, int paritybits);
void put_char(char a);       // skickar ett tecken
char receive_char(void);     // tar emot ett tecken
void receive_string(char*); // tar emot en sträng
void put_str(char*);         // skickar en sträng
void flushUSART(void);       // tömmer USART buffer
```

userdata.h

```
// Telefonnummer som skall ringas upp vid alarm.
#define TEL "0408249049;" // ; anger röstsamtal, annars datasamtal

// antal sekunder mellan det att ATD-kommandon skickas
#define SEKUNDER 150

// antal ringningar som utförs för varje ensklilt alarm, 0=oändligt.
#define MAX_RINGNINGAR 0
```

main.c

```
#include <avr\io.h>
#include <avr/interrupt.h>
#include "alarm.h"
#include "serial.h"
#include "userdata.h"
#include <avr/sleep.h>

int main(void)
{
/*************************************************************************
    INITIERAR REGISTER OCH STÄLLER IN OLIKA SAKER
*************************************************************************/

// Ställer in portriktingarna
    DDRA = 0xff;                // port A = utgångar
    DDRB = 0x00;                // port B ingångar

// Initierar portarna
    PORTA = 0x00;               // port A låg
    PORTB = 0xff;               // pull-up motstånden aktiv

// Ställer in avbrottsregistren (INT0/INT2)
    // INT0,INT2 TRIGGAS på rising edge
    EICRA |= (1 << ISC01) | (1 << ISC00) | (1 << ISC21) | (1 << ISC20);
    // nollar bort interruptflaggor
    EIFR |= (1 << INTF2) | (1 << INTF1) | (1 << INTF0);
    // TILLÅTER externa avbrott på pin INT0 och INT2!
    EIMSK |= (1 << INT2) | (1 << INT0);

// Ställer in 16 bit timer1
    // ställer in CTC-mode (Clear Timer on Compare)
    TCCR1B |= (1 << WGM12);
    // 7813 tar 1 sekund att räkna till med 7,8125kHz
    OCR1A = 7813;
    // tillåter compare-match-A-avbrott på timer1!
    TIMSK1 |= (1 << OCIE1A);

// Ställer in sleep mode
    SMCR |= (1 << SE);  // aktiverar default sleep mode (Idle)
```

```
// Initierar seriekommunikationskretsen USART0
    // baurate=9600, 8 databitar, 1 stopbit, ingen paritetsbit
    init_USART(9600);


    sei(); // Global flagga - tillåter härmed avbrott


/*****************************************************************************
    KÖR PROGRAMMET
*****************************************************************************/

    triggat_alarm = 0;// initierar den globala alarmflaggan
    ringningsnummer = 0;// initierar räknaren för antalet uppringningar

    PORTA |= (1 << PA1);     // Tänder aktivitetsLED
    put_str("AT");       // skickar AT kommandot "AT" för att anpassa...
                         // GSM modemet till "rätt" baud rate.
    put_char('\r');      // avslutar med \r
    flushUSART();            // tömmer USART0's buffer

    while(1)
        {
            PORTA &= ~(1 << PA1); // släcker aktivitetsLED
            sleep_cpu();         // aktiverar sleep-mode (idle)
        }

return 0;
}
```

alarm.c

```
#include "alarm.h"
#include <avr\io.h>

//*******************************************************
//
//  Funktion som kvitterar och nollställer alarmet...
//
//*******************************************************
void kvittering(void)
{
    PORTA = 0x02;        // tänder aktivitetsLED
                         // släcker alarmLEDen

    triggat_alarm = 0;   // nollar alarmflaggan
    TCCR1B = 0;          // stänger av timer0
    TCNT1 = 0;           // nollställer timer0

    ringningsnummer = 0;// nollar uppringningsräknaren
}
```

AT.c

```
#include "serial.h"
#include "userdata.h"

//*******************************************************
//                                                      *
//  Funktion som ringer upp en telefonnummer            *
//                                                      *
//*******************************************************
void ring(void)
{
    put_str("ATD"); // ATD, kommandot för att ringa.
    put_str(TEL);   // TEL = telefonnumret i userdata.h
    put_char('\r');
}
```

```
//*******************************************************
//                                                      *
//  Funktion som avbryter pågående samtal               *
//                                                      *
//*******************************************************
void hangup(void)
{
    put_str("ATH"); // ATH, Hang-up, avslutar samtalet
    put_char('\r');
}
```

serial.c

```
#include <avr\io.h>
#include <stdlib.h>
#include "serial.h"

//*******************************************************
//                                                      *
//  Funktion som initierar USART0                       *
//                                                      *
//*******************************************************
void init_USART(int baudrate)
{
    // aktiverar USART0 Rx och Tx
    UCSR0B |= (1 << RXEN0) | (1 << TXEN0);
    // Använder 8-bitars dataord
    UCSR0C |= (1 << UCSZ00) | (1 << UCSZ01);

    // Lagrar 8 lägre bitarna av baud rate värdet
    UBRR0L = (((F_CPU / (baudrate * 16UL))) - 1);
    // Högershiftar 8 steg och lagrar 8 högre bitarna
    UBRR0H = ((((F_CPU / (baudrate * 16UL))) - 1) >> 8);

    // Rx Complete Interupt Enable, TILLÅTER avbrott
    // för mottaget tecken på USART0
    UCSR0B |= (1 << RXCIE0);
}
//*******************************************************
//                                                      *
//  Funktion som skickar en byte på USART'en            *
//                                                      *
//*******************************************************
void put_char(char a)
{
        while ((UCSR0A & (1 << UDRE0)) == 0)
        {
            // väntar på att UDR blir klar
            // för att ta emot tecken.
        }

        UDR0=a; // skickar ut char a på serieporten
}


//*******************************************************
//                                                      *
//  Funktion som läser av ett tecken från               *
//  USART'ens buffer                                    *
//                                                      *
//*******************************************************
char receive_char(void)
{
        // RXC0 (bit7) på UCSR0A registret blir
        // 1 då data finns att tas emot!

        // Gör inget tills data har mottagits
        // och är klar att läsas från UDR0
        while ((UCSR0A & (1 << RXC0)) == 0) {};
        // läser av tecknet
        return UDR0;
}
```

```
//*****************************************************
//                                                     *
//  Funktion som skickar en sträng på USART'en         *
//                                                     *
//*****************************************************
void put_str(char a[])
{
    // medan tecknet som charpekaren a pekar på är != \0
    while (*a != '\0')
    {
        put_char(*a);// skicka ut det tecknet på USARTen
        a++;         // pekar på nästa element
    }
}

//*****************************************************
//                                                     *
//  Funktion som läser en hel sträng från USART Rx     *
//                                                     *
//*****************************************************
// tar emot en pekare till första elementet i en charvektor
void receive_string(char* str)
{
    // medan det värde som str pekar på tilldelas värdet
    // som finns på USART buffern och det värdet inte är
    // tecknet \r så görs inget.
    // OBS! detta lägger till \r som sista tecken!
    while ((*str++ = receive_char()) != '\r')
    {

        // Här behöver while-slingan inte utföra något
        // då värdet tilldelas i vilkoret för slingan.
    }


    // str pekar nu på positionen efter \r.
    // avslutar den mottagna strängen med \0.
    *str = '\0';
}
//*****************************************************
//                                                     *
//  Funktion som flushar USART                         *
//                                                     *
//*****************************************************
void flushUSART(void)
{
    char dummy;

    // Så länge som RXC0 är ett finns något att ta emot
    while(UCSR0A & (1 << RXC0))
    {
        // tar emot tecken till en dummychar.
        dummy = receive_char();
    }
}
```

serial.c

```
#include "userdata.h"
#include "alarm.h"
#include "AT.h"
#include "serial.h"
#include <avr/interrupt.h>
#include <util/delay.h>
#include <string.h>
```

```
//*****************************************
//                                        *
// Avbrott för mottagen data på USART0    *
//                                        *
//*****************************************
ISR(USART0_RX_vect)
{
    //skapar en charvektor
    char str[15];
    //lagrar mottagen sträng till den
    receive_string(str);
    // jämför om inkommande sträng är lika med "RING"
    if (strcmp("RING\r", str) == 0)
    {
        // OM alarmet är triggat
        if(triggat_alarm == 1)
        {
            // körs kvitteringsproceduren
            kvittering();
            // väntar två signaler
            _delay_ms(6000);
            // och avslutar sedan samtalet
            hangup();
        }
        else// ANNARS
        {
            // avslutas samtalet omedelbart
            hangup();
        }
    }
}
//*************************************************
//                                                *
//  Avbrott som körs när alarmsignal kommer       *
//                                                *
//*************************************************
ISR(INT2_vect)  // alarmsignalen kommer in på pin INT2
{

    PORTA |= (1 << PA1);    // tänder aktivitets LED'en
    // kollar om alarmet inte redan är triggat
    if(triggat_alarm == 0)
    {
        PORTA |= (1 << PA0);// tänder alarm LED'en
        triggat_alarm = 1;  // triggar alarmet
        ring();             // ringer upp ägaren
        // startar timer1
        TCCR1B |= (1 << CS12) | (1 << CS10);
    }
}


//*********************************************
//                                            *
//  Avbrott som körs vid manuell kvittering   *
//                                            *
//*********************************************
ISR(INT0_vect) // Kvitteringssignalen kommer på INT0
{
    if(triggat_alarm == 1)
    {
        kvittering();// kör kvitteringsproceduren
        hangup();   // avslutar sedan samtalet
    }
}
```

```
//*************************************************
//                                                 *
//  Avbrott som körs vid timer1 compare match A    *
//                                                 *
//*************************************************
ISR(TIMER1_COMPA_vect)
{
    static unsigned int tiden = 0;

    // SEKUNDER är önskat antal sekunder...
    //  mellan uppringningarna.
    if(tiden >= SEKUNDER)
    {
        // MAX_RINGNINGAR är maximala antalet
        // uppringningar per alarm.
        tiden=0;    // nollställer tidräknaren
        // ska alltid ringa om antal_ringningar är 0
        if(MAX_RINGNINGAR == 0)
        {
            ring();
        }

        // Kontrollerar villkoren för antalet
        //uppringningar och sekunder mellan uppringningar.
        if((MAX_RINGNINGAR > 0) &&
             (ringningsnummer <= MAX_RINGNINGAR))
        {
            ring();
            ringningsnummer++;
        }
    }
    else
    {
        tiden++;
    }
}
```