

Sommarstugekoll
Digital Konstruktion EDA234
Grupp 2

Fredrik Brosser, Karl Buchka, Andreas Henriksson, Johan Wolgers

Chalmers Tekniska Högskola

frebro @ student.chalmers.se
karlbu @ student.chalmers.se
henriksa @ student.chalmers.se
wolgers @ student.chalmers.se

12 december 2011

Sammanfattning

Ett system för temperaturavläsning och relästyrning beskrivs genom denna rapport. Systemet är åtkomligt och styrbart via telefon genom DTMF (Dual Tone Multiple Frequency). Systemets funktion består i att informera användaren om aktuella temperaturer från två temperatursensorer samt ge användaren möjlighet att via telefon styra utgångarna (till/från) för exempelvis värmeelement eller belysning. Styrning och återkoppling utbytes genom en telefonlinje, POTS (Plain Old Telephone Service) med hjälp av DTMF.

En tänkt tillämpning för produkten:

- * Systemet inkopplas innan avfärd från sommarstugan och befinner sig i viloläge.
- * Användaren kan då denne önskar ringa upp systemet som på anmodan returnerar temperatur från någon av temperaturgivarngita.
- * Användaren kan sedan genom sin knapptelefon aktivera eller avaktivera någon av utgångarna för att på så sätt slå till värme, belysning eller vad som anslutits till funktionsutgångarna.

Abstract

The system described in this report is an automated domestic temperature monitoring system, accessible and controllable via telephone. The main function of the system is to inform the user of the current temperatures at the points of measurement, and also to give the user remote binary control (on/off) of a number of external functions. These functions could for example be heating systems, radiators, or lighting. Information- and control data is exchanged via a standard analog telephone connection, POTS (Plain Old Telephone Service) using DTMF (Dual Tone, Multiple Frequency).

A specific, practical application is of the concerned holiday home owner wanting to monitor and control the temperature in his or her vacation property. Using Sommarstugekoll it is possible to keep heating expenditures to a minimum while at the same time reducing the risk of temperature related property damage such as burst water pipes.

Should the owner wish to visit the property, he or she needs only to dial the house telephone prior to arrival and instruct the system to raise the interior temperature to a comfortable level.

Innehåll

1	Introduktion	1
2	Kravspecifikation och systembeskrivning	2
2.1	Tolkning och definition av kravspecifikation	2
2.2	Uppdelning	2
2.3	Arbetsflöde	4
3	Funktionsenheter	5
3.1	Dataväg	5
3.2	Styrenhet	5
3.2.1	Uppbyggnad	6
3.3	DTMF-modul	7
3.3.1	Initiering	7
3.3.2	Läsning	7
3.4	Ljudmodul	7
3.4.1	Adressering och uppspelning	7
3.5	Temperaturmodul	8
3.5.1	Entrådsbuss	8
3.5.2	Läscykel	8
3.5.3	Uppbyggnad	10
3.5.4	Buffer/MUX	11
3.5.5	Räknare	11
3.6	Extern funktionsstyrning	11
3.7	Användargränssnittshantering	11
4	Hårdvara	12
4.1	ISD2560P	12
4.2	DS18S20	12
4.3	MT8880C	13
5	Tillståndsmaskiner	13
5.1	Styrenhet	14
5.2	Temperaturmodul	16
5.3	DTMF-Modul	18
5.4	Ljudmodul	20
6	Felanalys	22
	Appendix	23
A	Användarmanual och Gränssnitt	23
B	Blockschema	25

C	Komponentlista	29
D	Kretsschema	30
E	Kretskortslayout	31
F	Timingdiagram	32
G	Signallista	33
H	Arbetsfördelning	34
I	Datablad	35
J	Programlistningar	36

1 Introduktion

Konstruktionsprojektet utförs inom kursen “Digital Konstruktion, EDA234” vid Chalmers Tekniska Högskola. Uppgiften består att inom gruppen om 4 personer konstruera och dokumentera ett digitalt system utifrån en specifikation, med fokus på huvudfunktionalitet i det digitala området. För en beskrivning av arbetsfördelningen inom gruppen, se Appendix H. Utvecklingen sker på ett färdigt utvecklingskort vilket sedan kompletteras med externa kretsar och programmeras för att nå kravspecifikationen. Logik-kretsen som används är en Xilinx XC9572XL CPLD, och utvecklingen sker i huvudsak med VHDL i Xilinx ISE-miljön samt i ModelSim för simuleringar. Rapporten är uppdelad i abstraktionslager med fördjupningar i de senare avsnitten.

En handhavandeinstruktion återfinns bifogad i Appendix A.

2 Kravspecifikation och systembeskrivning

Systemet består av en styrenhet placerad på två CPLD med anslutning till telefonnätet. Styrenheten har med hjälp av en DTMF avkodare möjlighet att tolka användarens telefonknapptryckningar på distans. Vidare kan användaren genom dessa knapptryckningar skicka signal för att hämta värde från någon av de två temperaturgivarna anslutna med entrådsbus. En röst enhet läser sedan upp ett förinspelat meddelande för respektive temperatur. Vidare kan användaren också starta eller stoppa någon av de funktionsutgångar som finns till hands, användaren får då akustisk återkoppling om aktuell status för funktionsutgången via samma ljudmodul.

Funktionsutgångarna kan också styras lokalt med bifogade knappsetsen. Aktuell status för funktionsutgångarna visas med lysdioder.

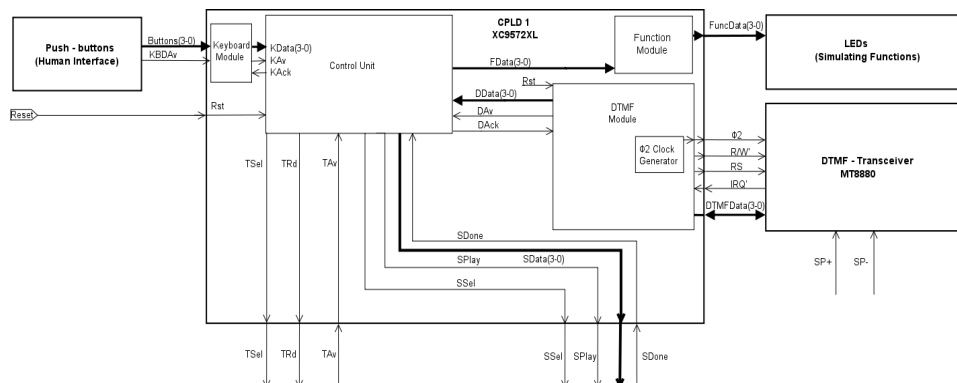
2.1 Tolkning och definition av kravspecifikation

Inom det tilltänkta användningsområdet, ett fritidshus med POTS-linje som under större delen av året endast nyttjas tillfälligtvis, är det rimligt med en minsta temperatur om $5 - 10^{\circ}\text{C}$ då ingen vistas i huset. Vidare bör det vara önskvärt att ha möjlighet att slå till normal värme innan ett besök, kunna kontrollera att det blivit varmt samt att kontrollera att temperaturen är över 0°C . Systemet kan därför tolka en temperatur om $\pm 32^{\circ}\text{C}$ från givarna med en upplösning om $\pm 1^{\circ}\text{C}$. Senast avlästa temperatur visas också binärt på LED-displayen. Systemet matas med en spänning om 5 V DC , då systemet är till indikeras detta med en lysdiod.

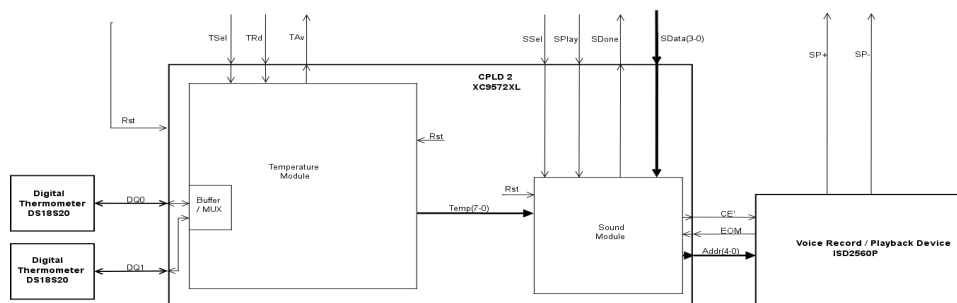
2.2 Uppdelning

Systemet är uppdelat i ett antal delblock (moduler) och är konstruerat för att vara så modulärt som möjligt. I tabell 1 nedan finns en beskrivning över hur de olika modulerna är uppdelade på de två CPLD:erna. Varje modul är mer utförligt beskriven under sektion 3 Funktionsenheter.

I blockschema som återfinns i figur 1 och 2 visas respektive modul och uppdelningen på två CPLD samt dess kommunikationsvägar. För större figurer, se Appendix B.



Figur 1: Blockdiagram för CPLD1



Figur 2: Blockdiagram för CPLD2

Tabell 1: Respektive CPLDs moduler och uppgifter

CPLD1

Modul	Funktion
Styrenhet	Samordnar systemets funktion
DTMF-modul	Tar emot DTMF-signaler via POTS
Funktionsmodul	Hanterar funktionsutgångarna
Knappmodul	Hanterar tryckningar på knappsatsen

CPLD2

Modul	Funktion
Temperaturmodul	Initierar, läser och presenterar temperatur
Ljudmodul	Hanterar styrning av extern ljudkrets

Systemet är uppdelat enligt "dataväg-styrenhet-modellen", där design-principen går ut på att skilja styrsignaler och dataflödeskontroll (styrenhe-

ten) från datan som forslas genom systemet (datavägen). Systemet skickar temperaturdata från temperaturmodulen till ljudmodulen samt adressdata till ljudlagringsenheten från ljudmodulen. Detta flöde kontrolleras från styrenheten via enkla styrsignaler. Viss data, i form av indata från användargränssnitt och styrsignaler till funktionsutgångar, passerar dock genom styrenheten.

2.3 Arbetsflöde

I sitt viloläge väntar systemet på inkommande samtal. Då samtal mottages börjar systemet med att läsa upp ett antal ljudklipp med aktuella temperaturer, och erbjuder sedan användaren möjligheten till att styra de externa funktionerna. Telefonen skickar ut DTMF-signaler, som läses och avkodas av DTMF-transceivern, MT8880C, vilken avkodar tvåtonssignalerna för att identifiera nedtryckt knapp och sända ut denna på databussen till DTMF-modulen.

Då användaren trycker ned en knapp kontrollerar styrenheten om knapptryckningen motsvarar ett funktionskommando (funktion till eller från), i så fall sätts statusen för respektive funktionsutgång, annars återgår systemet till att invänta nytt kommando.

3 Funktionsenheter

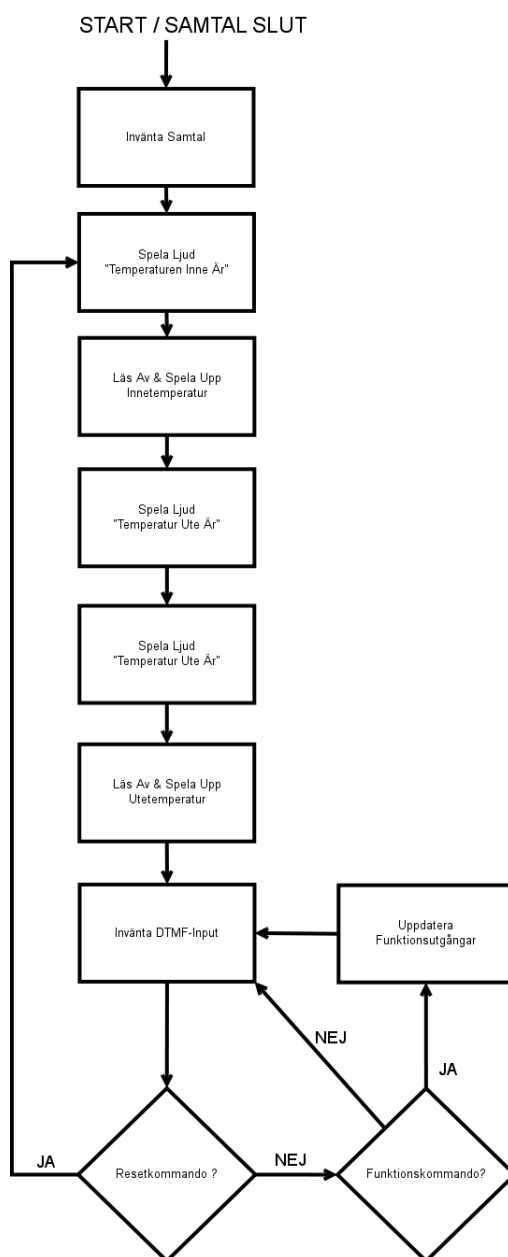
Systemet är som tidigare nämnt uppdelat i sex funktionsenheter: funktionsmodul, knappsatsmodul, styrenhet, DTMF-modul, ljudmodul och temperaturmodul. Dessa presenteras närmare i respektive avsnitt nedan.

3.1 Dataväg

Temperaturdaten avläses från temperatursensorerna och skickas till temperaturmodulen. Denna skickar sedan datan till ljudmodulen, för vidare uppläsning via ljudlagringskretsen, ISD2560P. Denna data skickas aldrig genom, men kontrolleras av, styrenheten. Den rena datavägen i systemet är alltså här begränsad till CPLD2.

3.2 Styrenhet

Styrenheten kontrollerar och ger styrsignaler för att hantera de övriga systemdelarna. Syftet är att ge en lätt överblick över hela körcykeln, där styrenheten har kontroll över funktionerna via en rad styr-, available- och acknowledge-sig-naler. Styrenheten får även indata från knappsats och DTMF-mottagare via respektive modul som presenterar data för styrenheten. Denna indata är skild från datavägen (temperaturinformation), och meddelar styrenheten om nästföljande steg, dvs. reagera i enlighet med användarsignaler. Se figur 3 för flödesschema.



Figur 3: Högnivå-flödesdiagram för styrenhet

3.2.1 Uppbyggnad

Styrenheten är i sin tur modulärt uppbyggd, och har ett väl avgränsat interface mot varje annan delmodul, men fungerar samtidigt som en samordnare mellan de olika modulerna. För en ingående grafisk beskrivning av den bakomliggande tillståndsmaskinen, se sektion 5 Tillståndsmaskiner.

3.3 DTMF-modul

DTMF-modulen i CPLD1 är ansvarig för att hantera kommunikation med telefonen och därmed också med användaren. När användaren trycker ner en knapp på telefonen genereras en DTMF-signal som uppfattas och avkodas av DTMF-modulen vilken sedan presenterar den mottagna indatan för styrenheten.

3.3.1 Initiering

DTMF-modulen måste genomgå en initieringssekvens minst 100 ms efter spänningspåslag. Detta sker med hjälp av en extern tryckknapp, som beskrivet i användarmanualen. Under hela initieringen är Chip Select satt låg och RS0 satt hög. Initieringscykeln börjar med att DTMF-modulen läser statusregistret och ger kommandon genom att skriva till kontrollregistret på MT8880C-kretsen enligt ett i databladet givet mönster. Se även sektion 4.3 MT8880C under Hårdvara för vidare beskrivning. DTMF-modulen är ansvarig för att ge ut rätt signaler under initieringsfasen, vilket görs med hjälp av en tillståndsmaskin. Denna finns beskriven i sektion 5 Tillståndsmaskiner.

3.3.2 Läsning

Då initieringen av MT8880C är färdigställd går DTMF-modulen över i ett läsläge, där den kontinuerligt lyssnar efter insignaler från MT8880C-kretsen. När giltig indata detekteras läses den och läggs ut på den interna DTMF-databussen till styrenheten, varpå DTMF-modulen signalerar att ny data avkodats till styrenheten. DTMF-modulen väntar sedan på en bekräftelse-signal (dAck) från styrenheten. Läsningen sker genom att DTMF-modulen väntar på signal på avbrottsutgången (irq) vilket betyder att en ny DTMF-signal detekterats och avkodats från telefonlinjen och lagts ut på databussen från MT8880C-kretsen.

3.4 Ljudmodul

Ljudmodulen styr den externa ljudlagringskretsen och signalerar till styrenheten när en uppspelning är färdig. Genom att ställa om värdet av c/t signalen kan styrenheten begära en vanlig ljuduppspelning eller en temperaturuppspelning.

3.4.1 Adressering och uppspelning

När styrenheten begär en temperaturuppspelning så börjar ljudmodulen att läsa av temperaturmodulens databuss. Om temperaturen är giltig (dvs. innanför det giltiga mätspannet) så skickar ljudmodulen ut de 6 minst signi-

fikanta bitarna av temperaturen direkt till ljudkretsen och triggas en uppspelning. Ljudkretsen är programmerad på sådant vis att temperaturbussens värde är mappat 1:1 till det korrekta ljudklippet. Om en overflow har skett så ställs temperaturen helt enkelt till det högsta giltiga värdet och uppspelningen fortsätter. Efter att själva temperaturen är färdiguppspelad så granskar ljudmodulen åter igen temperaturbussen och spelar upp ”minusgrader” eller ”plusgrader” beroende på vilken sida nollstreckets man befinner sig på. Därefter signaleras styrenheten att uppspelningen är färdig.

Vid en vanlig uppspelning använder ljudmodulen en 4-bitars adress kommande från styrenheten. Innan uppspelning förlängs denna till 6 bitar genom att lägga till en ny statisk ”Most Significant Bit” (MSB) och en ny statisk ”Least Significant Bit” (LSB). Adressen läggs på så sätt in mellan två nya bitar. Anledningen till förlängningen är att ljudklippet som styrenheten begär kräver mer lagringsutrymme på ljudkretsen än vad temperaturerna gör. För mer detaljerad förklaring, se sektion 4.1 ISD2560P under Hårdvara.

3.5 Temperaturmodul

Temperaturmodulen är ansvarig för att hantera seriekommunikationen med temperatursensorerna DS18S20, via entrådsbussarna, samt att ge ut de lästa temperaturerna i tecken-belopp-format, där den mest signifikanta biten ger tecknet (positiv eller negativ) och efterföljande bitar temperaturdaten binärt. Temperaturerna skickas via en intern databuss till ljudmodulen, i syfte att låta den i sin tur spela upp de avlästa temperaturen för användaren. För vidare beskrivning av temperatursensorkretsen, se sektion 4.2 DS18S20 under Hårdvara.

3.5.1 Entrådsbuss

Temperaturmodulen kommunicerar seriellt med temperatursensorn över en entrådsbuss. Entrådsbussen är ansluten till matningsspänning genom ett pull-up-motstånd. Bussens två ändar är anslutna till CPLD och sensor, respektive. När information skickas över bussen drar den sändande enheten bussen låg genom att driva den med en stark logisk nolla.

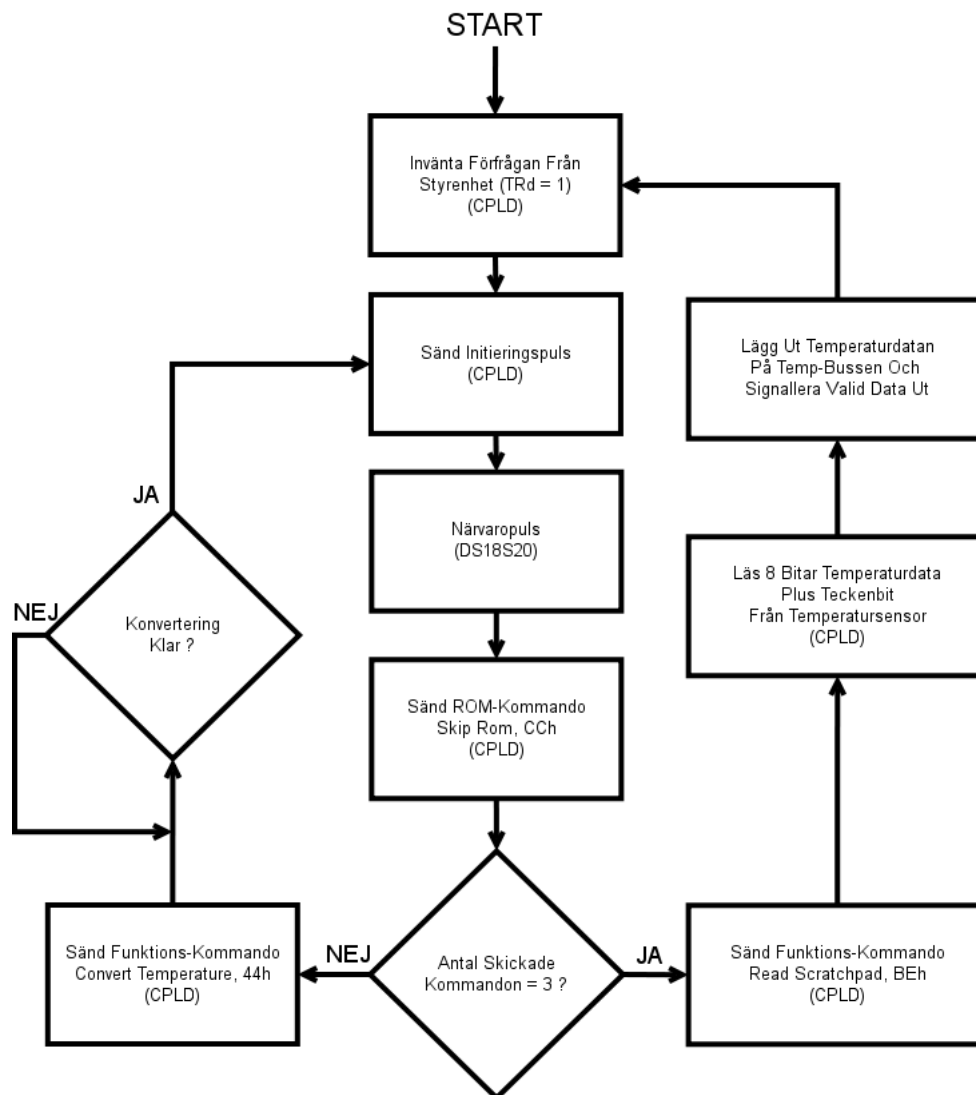
3.5.2 Läscykel

En läscykel består av fyra steg: initiering, kommando, läsning och viloläge, se även figur 4 för en flödesdiagram-beskrivning av en hel läscykel.

Initieringen består av att mastern driver bussen låg i $512\mu s$ och sedan släpper den. Temperatursensorn svarar då med en närvaropuls genom att driva bussen låg i $106\mu s$.

Då mastern detekterat närvaropulsen börjar den överföra ett ROM-kommando (Skip ROM, 0xCC), följt av en kort återhämtningsslucka och

sedan ett funktionskommando (Convert Temperature, 0x44). “Skip-ROM-kommandot” används i det här systemet, endast en temperatursensor används per entrådsbuss, därmed finns inget behov av att kunna adressera specifika sensorer på bussen. Då “Convert Temperature”-kommandot ges gör DS18S20-kretsen en temperaturkonvertering, för att sedan spara den lästa temperaturdaten i ett internt minne. Under konverteringsperioden (upp till 750 ms) kan mastern göra polling av temperatursensorns status. Detta innebär att mastern kontinuerligt sänder förfrågningar genom att dra bussen låg i en kort period (4 μ s). Temperatursensorn svarar med en nolla då konvertering sker, och en etta då konverteringen är klar. Då mastern ser att konverteringen är klar, initieras temperatursensorn om och ytterligare ett ROM-funktions-kommandopar överförs. Dessa är 0xCC (Skip ROM) följt av 0xBE (Read Scratchpad), respektive. Temperatursensorn är då redo att överföra temperaturdata till mastern från sitt interna minne. Då den ger “Read Scratchpad”-kommandot börjar DS18S20 sända innehållet i sitt minne över bussen. Mastern går då in i läsningsläget och börjar sampla data från bussen genom att driva bussen låg, släppa den och sampla efter 4 μ s. Mastern samplar de första åtta bitarna som sänds av DS18S20, sedan en ytterligare, nionde bit. Den sista biten utgör en tecken-bit för att avgöra om temperaturen är positiv eller negativ. Då den nionde biten data lästs ger mastern på nytt en initieringspuls för att avbryta läsningen. Den nyligen lästa temperaturen placeras på den interna temperaturbussen och temperaturmodulen signalerar till styrenheten att giltig data finns på bussen. För timingdiagram, se Appendix ??.



Figur 4: Högnivå-flödesdiagram för temperaturläscykel

3.5.3 Uppbyggnad

Temperaturmodulen är baserad på en tillståndsmaskin, understödd av en intern buffer/multiplexer (MUX) samt ett antal interna räknare för att generera de nödvändiga tidsfördröjningspulserna. För en grafisk beskrivning av den interna tillståndsmaskinen som används av temperaturmodulen, se sektion 5 Tillståndsmaskiner.

3.5.4 Buffer/MUX

Buffern är en “tri-state-buffer” med en “enable”-signal. Buffer/MUX-modulen är direkt ansluten till de två DS18S20-temperatursensorerna som använder entrådsbusskommunikationen. Multiplexern används för att välja mellan vilken av de två sensorerna som styrenheten vill kommunicera med.

3.5.5 Räknare

Internt använder temperaturmodulen ett antal räknare för att generera de nödvändiga timingpulserna och generera korrekta läs- och skrivluckor, se tabell 2.

Tabell 2: Interna räknare, temperaturmodul

Räknare		
Namn	Bitar	Beskrivning
cntInt	9	Genererar timing-pulser
ZC	4	Hanterar timing för skriv-luckor
Progress	2	Anger aktuellt kommando (0-3)
bitCnt	8	Anger aktuell bit för överföring/läsning

Se även VHDL-beskrivning i Appendix J.

3.6 Extern funktionsstyrning

Funktionsmodulen tar emot data från styrenheten och skickar vidare uppdateringarna på funktionsutgångarna som används för funktionerna. Sammanfattat fungerar den som ett mellansteg för att underlätta för styrenhetens funktionsutgångsstyrning, och kan vid behov byggas ut ytterligare för att implementera andra funktioner i systemet.

3.7 Användargränssnittshantering

Systemet har, förutom DTMF-kontroll, även ett lokalt knappsatsgränssnitt och en till/från-brytare för temperaturläsning. Dessa rådata måste formateras, avkodas och presenteras för styrenheten. Detta är knappsatsmodulens uppgift.

Om en knapp på knappsatsen trycks ned, skickas en “available”-signal (dAv) från knappsatsen till knappsatsmodulen, som sedan avkodar vilken knapp som tryckts ned, och presenterar detta för styrenheten genom att höja en “keyboard available” (kAv)-signal, vilket indikerar giltig data på knappsatsdatabussen (kData[3..0]). Genom att skicka en “acknowledgement”-signal (kAck) bekräftar styrenheten att den sett och mottagit datan från knappsatsen.

4 Hårdvara

Nedan presenteras hårdvaran som systemet är uppbyggt av, under respektive rubrik. Se även komponentlista i tabell 3, Appendix C samt kretsschema i figur 15, Appendix D. Pinlayouten för CPLD-kretsarna ges i kretsschemat.

4.1 ISD2560P

ISD2560P är en integrerad inspelnings- och uppspelningskrets från Futurlec, med plats för 60 sekunder ljud. Kretsen är inkopplad enligt ett modifierat förslagsschema i databladet och brukas i "Direct Addressing Mode". Kretsen adresseras med 10 bitar, vilket ger 600 giltiga adresser. Varje adressinkrementering flyttar uppspelningspekaren med 0.1 sekunder ($0.1 \text{ sekunder} \cdot 600 \text{ adresser} = 60 \text{ sekunder}$). Signalerna "chip enable" (ce), "end of message" (eom) och en 6-bitars adressbuss är anslutna till CPLDn. Uppspelningen styrs av den flanktriggade signalen ce som är aktiv låg. När denna får en puls latchas adressvärdena och en uppspelning påbörjas. Uppspelning avslutas med en annan (aktiv låg) puls på eom.

De 3 minst signifikanta adressbitarna och den mest signifikanta adressbiten är knutna till jord. Resterande 6 bitar styrs av CPLDn. Inkoppling på detta vis innebär ljudklippslängder i multiplar av 0.8 sekunder. Alla siffror som läses upp ryms i enskilda 0.8-sekundersblock och kan således adresseras utan vidare. Längre ljudklipp kräver aldrig mer än 1.6 sekunder och kan tilldelas block av dubbel längd genom att använda 5-bitarsadressering med en statisk LSB. Se även datablad i Appendix I.

4.2 DS18S20

Den temperatursensor som används är Maxim DS18S20, som ger temperaturmodulen möjlighet att läsa av temperaturen med nio bitars upplösning. Sensorerna som används i detta system drivs av en extern spänningskälla på +5V, och kommunicerar seriellt över en entrådsbuss. Seriekommunikationen baseras på att "bus-mastern" (med bus-mastern avses här CPLD1) initierar skriv- och läsluckor. Mellan varje skriv- eller läslucka finns en kort återhämtningsperiod. Temperatursensorn initieras genom att mastern driver bussen låg, vilket följs av att temperatursensorkretsen driver för att signalera närvaro. Efter initieringen väntar sensorkretsen på ett ROM-kommando från mastern, följt av ett funktionskommando. Varje sådant kommando är en byte långt och skickas som LSB-först. Mastern skickar en logisk nolla genom att driva bussen låg under hela skrivluckan. En etta skickas genom att mastern driver bussen låg under en kort del av skrivluckan och sedan släpper bussen under resten av skrivluckans längd.

Då mastern är klar med att skicka över ROM- och funktionskommando, kan (beroende på vilka kommandon som sändes) DS18S20-kretsen svara

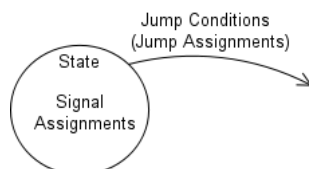
med aktuell data. På samma sätt som ovan måste mastern här initiera en läslucka. DS18S20-kretsen svarar på den allokerade läsluckan och överför etta eller nolla. Mastern kan då sampla bussen för att läsa av vad DS18S20-kretsen skickat. All data från temperatursensorn skickas som “LSB-first” och i 2-komplementsform. För exakta tidskrav, se datablad i Appendix I

4.3 MT8880C

Avkodning av inkommande DTMF-signaler avkodas med hjälp av MT8880C från Mitel. Telefonsignalen kopplas direkt in till DTMF-överföraren enligt specifikation för standarduppkoppling i datablad. Endast avkodningsfunktionen på kretsen används. Databuss, r/w-signal, irq och $\Phi 2$ är anslutna till styrenheten. “chip select” (ce) är satt konstant låg då denna krets använder bussen exklusivt. Före initiering skickas manuellt en initieringssekvens till kretsen minst 100 *ms* efter spänningspåslag. Detaljer om initieringscykeln finns i databladet för MT8880C, se Appendix I.

5 Tillståndsmaskiner

Nedan beskrivs modulernas tillståndsmaskiner. Varje tillståndsmaskin presenteras som en lista över tillstånd med förklaringar, samt som ett grafiskt tillståndsdigram. Diagrammen är uppbyggda enligt principen som visas i figur 5



Figur 5: Förklaring till tillståndsdigram

5.1 Styrenhet

Se figur 6

Reset: Alla signaler sätts låga.

Grundvärden: Utgångspunkten är att alla signaler behåller sina gamla värden om inget annat anges.

Viloläge (tillstånd 0):

0: Inväntar samtal.

Temperaturuppläsning (tillstånd 1-5):

3: Läser upp ljud "Temperaturen inne är".

4: Läser upp innetemperatur.

3: Läser upp ljud "Temperaturen ute är".

4: Läser upp utetemperatur.

5: Läser upp ljud "Funktionskommandotilldelning".

Kommando (tillstånd 6):

6: Inväntar funktionskommando eller avslutat samtal.

Funktionstilldelning (Tillstånd 7-11):

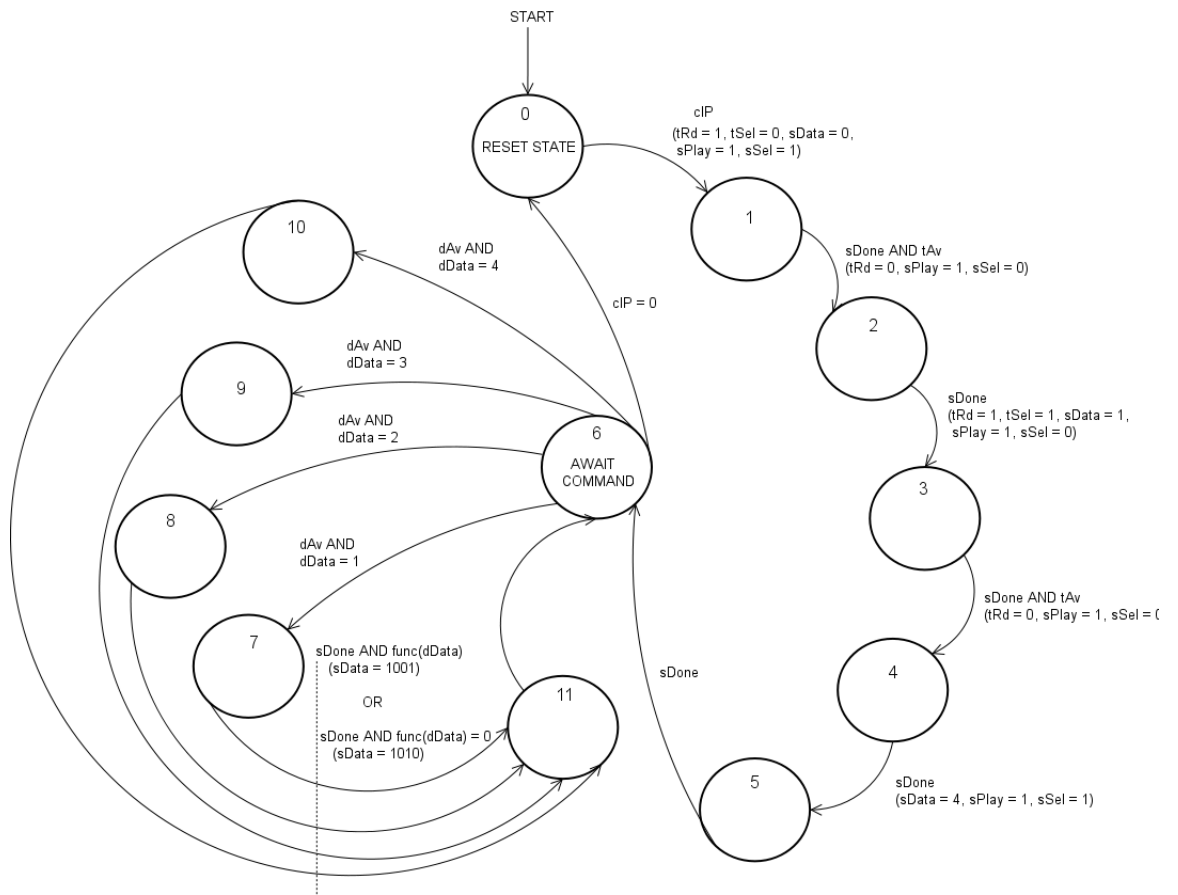
7: Uppdaterar/läser upp funktionsstatus för funktion 1.

8: Uppdaterar/läser upp funktionsstatus för funktion 2.

9: Uppdaterar/läser upp funktionsstatus för funktion 3.

10: Uppdaterar/läser upp funktionsstatus för funktion 4.

11: Läser upp funktionsstatus "till/från".



Figur 6: State machine-diagram för styrenhet

5.2 Temperaturmodul

Se figur 7.

Reset: Alla signaler och räknare sätts låga.

Grundvärden: Utgångspunkten är att alla signaler behåller sina gamla värden om inget annat anges.

Viloläge (tillstånd 0):

0: Viloläge och återställningspunkt. inväntar t_{Rd} från styrenheten.

Initialisering (tillstånd 1-3):

1: Fördröjningstillstånd, väntar på puls på $delayLong$ ($512\mu s$).

2: Mastern driver bussen låg i $512\mu s$ och sätter datavärdet till $0xCC$ (Skip ROM).

3: Mastern släpper bussen, DS18S20 skickar närvaropuls.

Kommandoöverföring (tillstånd 4-8):

4: Förberedelsestillstånd före sändning.

5: Huvudsändningstillståndet, mastern sänder bit enligt aktuellt räknarvärde.

6: Mellanbittillstånd, återhämtningstillstånd. Fortsätter om fler bitar ska skickas.

7: Avgör om DS18S20 är upptagen med att konvertera temperaturen. Om så, vänta tills klar.

8: Vägsälstillstånd. Om fler kommandon kvar att skicka, gå tillbaka, annars börja läsa.

Läsning (Tillstånd 9-15):

9: Förberedelsestillstånd före läsning.

10: Mastern initierar en läslucka genom att dra bussen låg i $4\mu s$.

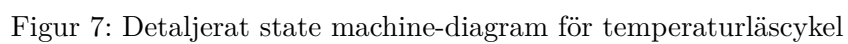
11: Mastern väntar ytterligare $4\mu s$ för att pull-up-motståndet ska få verka.

12: Mastern samplar bussen. Om DS18S20 skickar en nolla hålls bussen låg, annars inte.

13: Återhämtningstillstånd mellan samplingar.

14: Vägsälstillstånd. Om vi har fler bitar att läsa, gå tillbaka, annars gå vidare.

15: Läsning klar, lägg ut temperatur på buss och signalera giltig data. Gå tillbaka till 0.



5.3 DTMF-Modul

Se figur 8.

Reset: Alla signaler sätts låga.

Grundvärden: Utgångspunkten är att alla signaler är satta till 0 om inget annat anges.

Viloläge (tillstånd 0):

0: Viloläge och återställningspunkt, inväntar “sigInit”.

Initialisering (tillstånd 1-20):

1-3: Läs statusregister.

4-6: Skriver ‘0000’ till kontrollregister A.

7-8: Skriver ‘0000’ till kontrollregister A.

9-11: Skriver ‘1000’ till kontrollregister A.

12-14: Skriver ‘0000’ till kontrollregister B.

15-17: Läser statusregister.

18-20: Intialisering av Chip-interface.

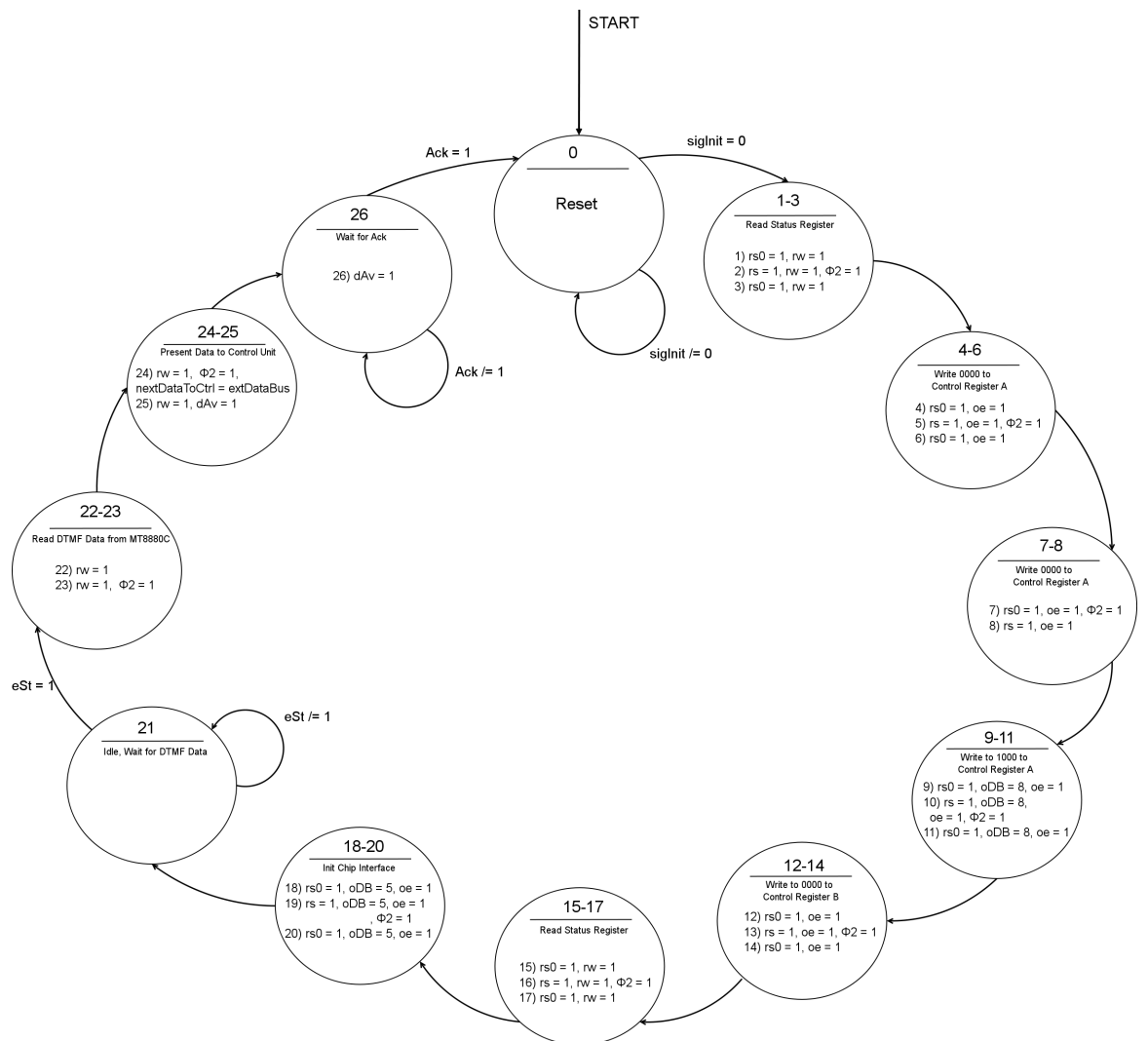
Läsning (Tillstånd 21-26):

21: Viloläge, inväntar giltig data från MT8880C.

22-23: Läs data från MT8880C.

24-25: Lägga ut data på intern databuss och signalera giltig data.

26: Inväntar bekräftelsesignal från styrenhet.



Figur 8: Förenklat state machine-diagram för DTMF-Modul

5.4 Ljudmodul

Se figur 9

Reset: Alla signaler återställs låga.

Grundvärden: Alla signaler låga, utom ce som är hög.

Viloläge (tillstånd S0):

S0: Inväntar play-signal.

Spela upp Ljud (C)

S1C: Sätt adress.

S2C: Spela upp ljud, vänta tills uppspelning är klar.

S3C: Signalera uppspelning klar till styrenhet.

Spela upp Temperatur

S1T: Sätt adress.

S2T: Spela upp temperatur, vänta tills uppspelning klar.

Spela upp Temperatur (vid overflow)

S1TO: Sätt adress.

S2TO: Spela upp temperatur, vänta tills uppspelning klar.

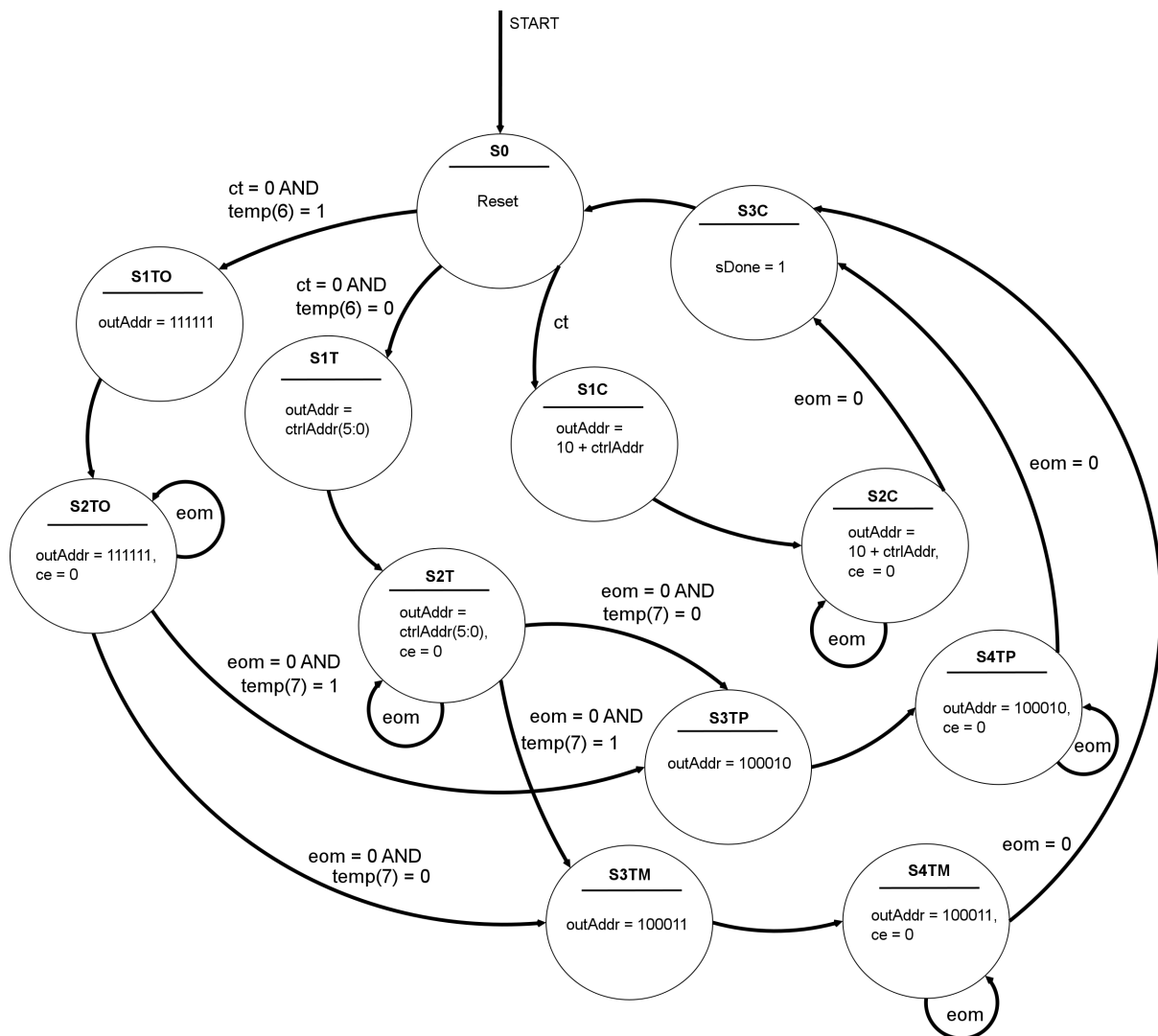
Spela upp Positiv/Negativ

S3TP: Sätt adress (positiv temperatur).

S3TM: Sätt adress (negativ temperatur).

S4TP: Spela upp positiv, vänta tills klar.

S3TM: Spela upp negativ, vänta tills klar.



Figur 9: State machine-diagram för ljudenhet

6 Felanalys

Då systemet hanterar avläsning av temperaturer och endast klarar ett begränsat temperaturintervall görs här en felanalys. För de tillämpningsområden systemet är konstruerat för (hemanvändare) är mätfelet dock för små för att ha betydelse; noggrannheten är större än vad behovet åkallar.

Temperatursensorerna arbetar med en temperaturupplösning på $0,5\text{ }^{\circ}\text{C}$ enligt datablad, se Appendix I. Temperaturen levereras på en absolut skala och sensorerna ska leverera en exakthet på $\pm 0,5\text{ }^{\circ}\text{C}$ i intervallet $-10\text{ }^{\circ}\text{C}$ till $+85\text{ }^{\circ}\text{C}$. Resultatet som presenteras på LED-displayen är hela det avlästa värdet i binär form, medan ljuduppspelningen inte presenterar mer än en grads upplösning. Systemet arbetar med ett temperaturspann på $\pm 32\text{ }^{\circ}\text{C}$. Temperaturer utanför intervallet representeras som högsta respektive lägsta värde.

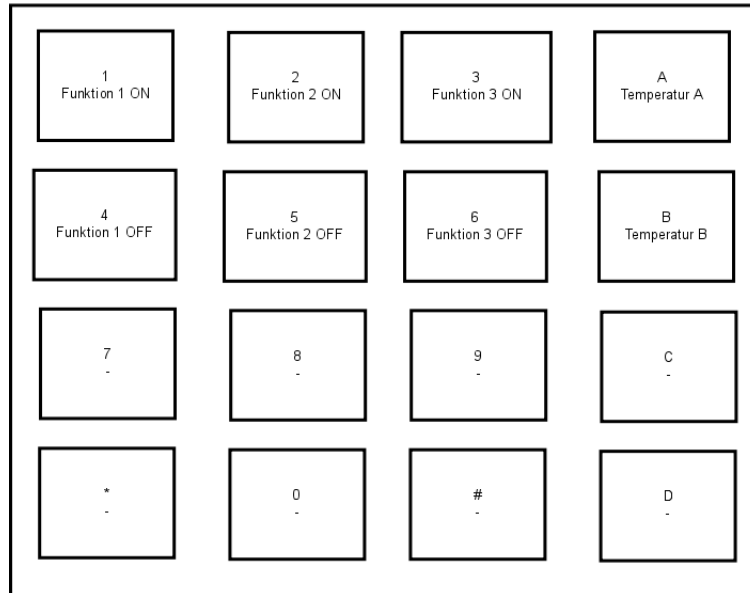
Appendix

A Användarmanual och Gränssnitt

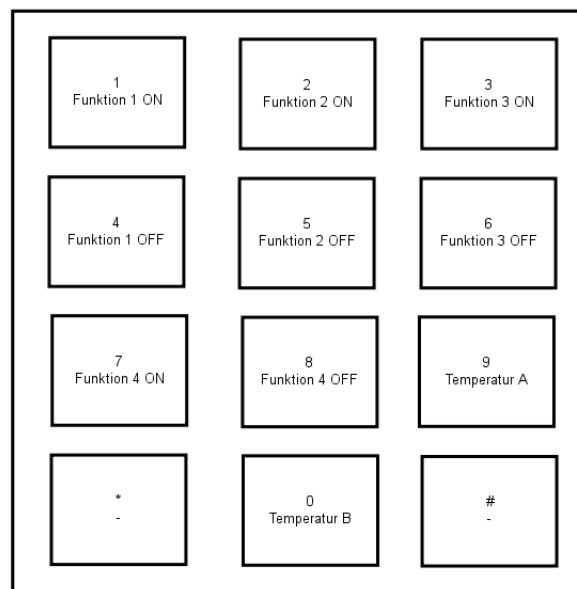
För att använda systemet, spänningssätt +5V DC mellan ingångarna “+5V” och “GND” och koppla in telefonlinje till DTMF-anslutningen. När systemet är startat ska det återställas (röd tryckknapp) och sedan initieras (svart tryckknapp). Efter detta är systemet redo att användas, kommando kan ges via telefon eller manuell knappsats, se figur ?? . När som helst kan systemet startas om helt genom att igen trycka på den röda tryckknappen. Det finns även en knapp för att slå till eller från kontinuerlig avläsning av temperatur. Då systemet blir uppringt läses temperaturen upp, sedan kan kommandon ges från användaren enligt nedan:

Knappsats		Telefon	
Knapp	Funktion	Knapp	Funktion
1	Funktion 1 TILL	1	Funktion 1 TILL
2	Funktion 2 TILL	2	Funktion 2 TILL
3	Funktion 3 TILL	3	Funktion 3 TILL
4	Funktion 1 FRÅN	4	Funktion 1 FRÅN
5	Funktion 2 FRÅN	5	Funktion 2 FRÅN
6	Funktion 3 FRÅN	6	Funktion 3 FRÅN
A	Visa Innetemperatur	9	Visa Innetemperatur
B	Visa Utetemperatur	0	Visa Utetemperatur

KNAPPSATS

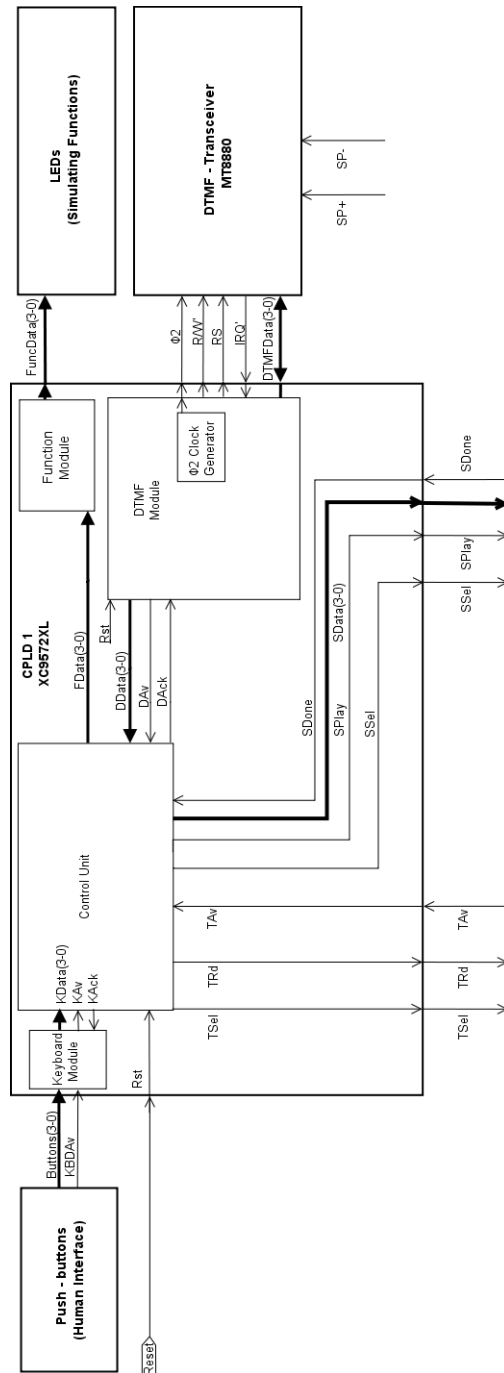


TELEFONKNAPPAR

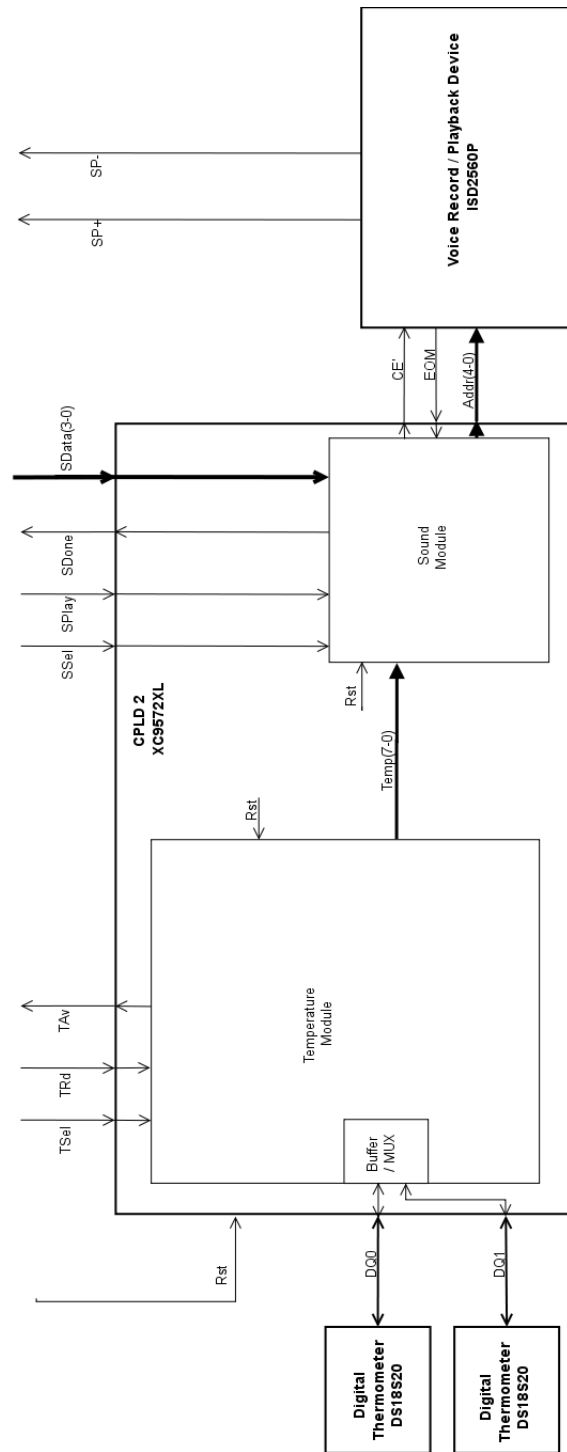


Figur 10: Knapplayout, knappsats resp. telefon

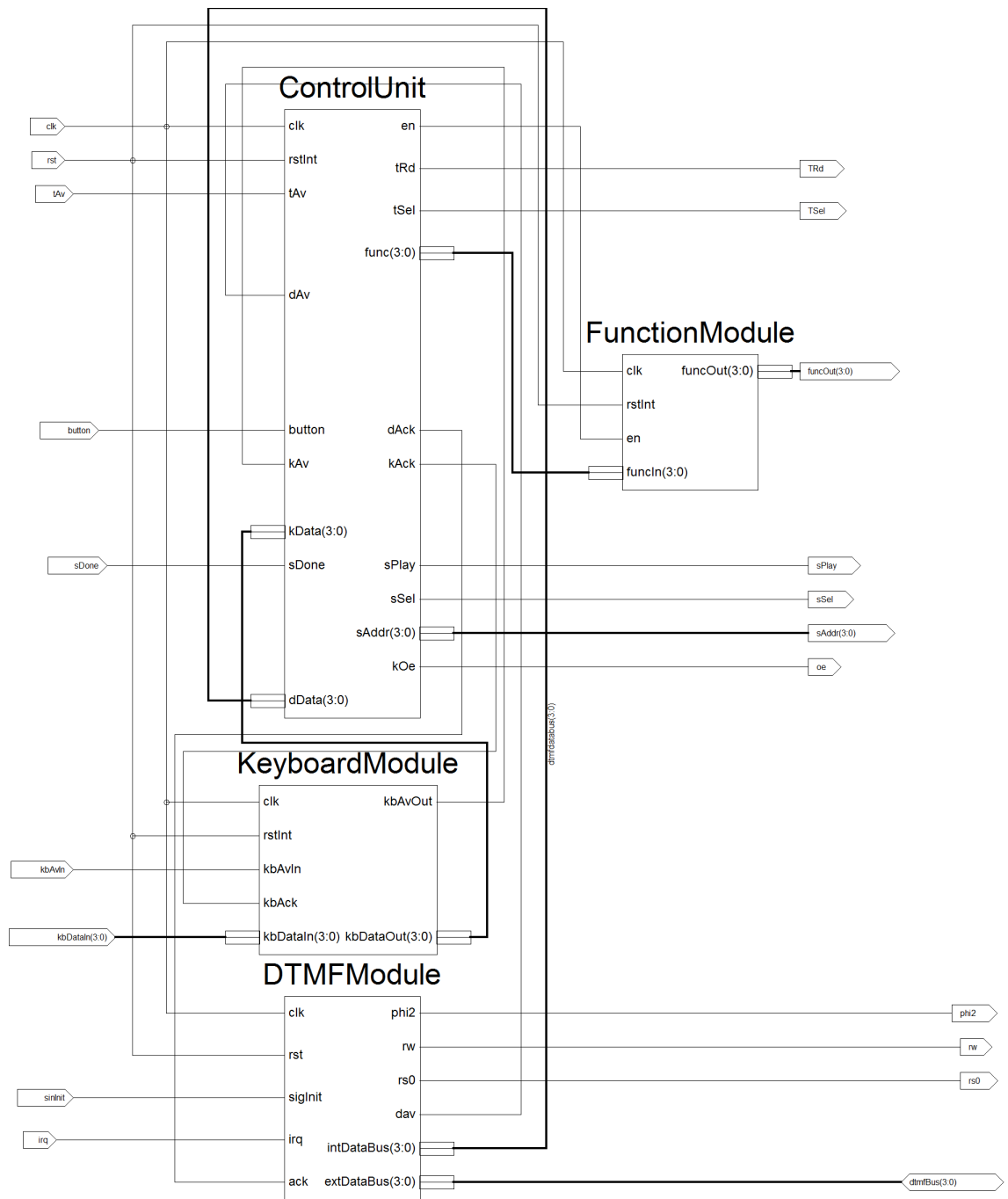
B Blockscha



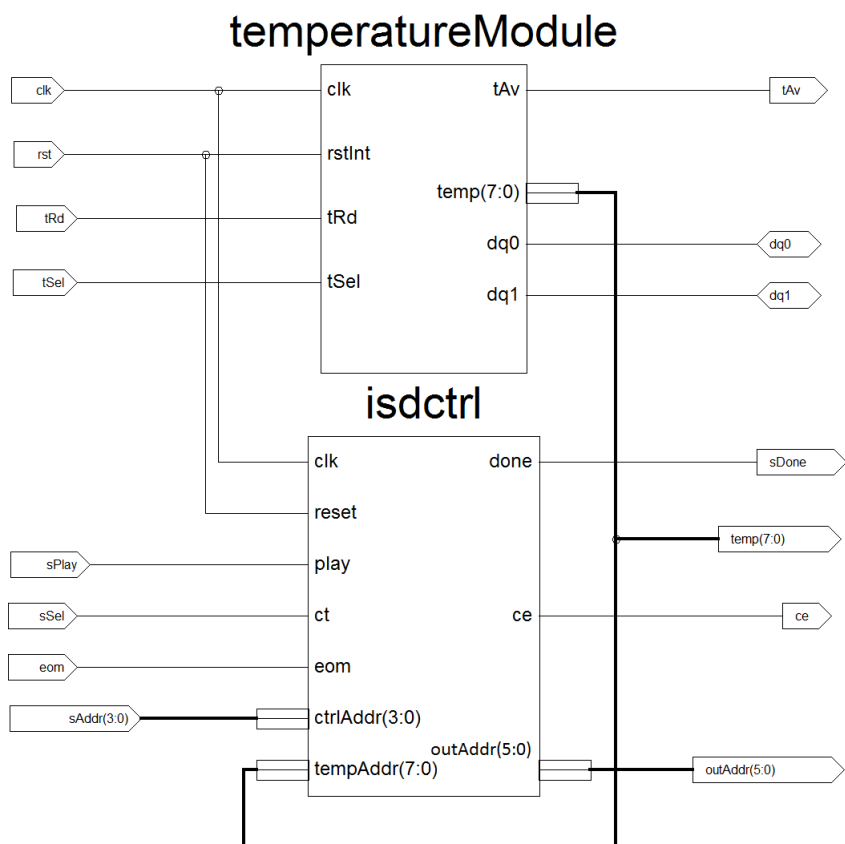
Figur 11: Blockscha (CPLD1)



Figur 12: Blockschema (CPLD2)



Figur 13: Internt blockschema, moduler (CPLD1)



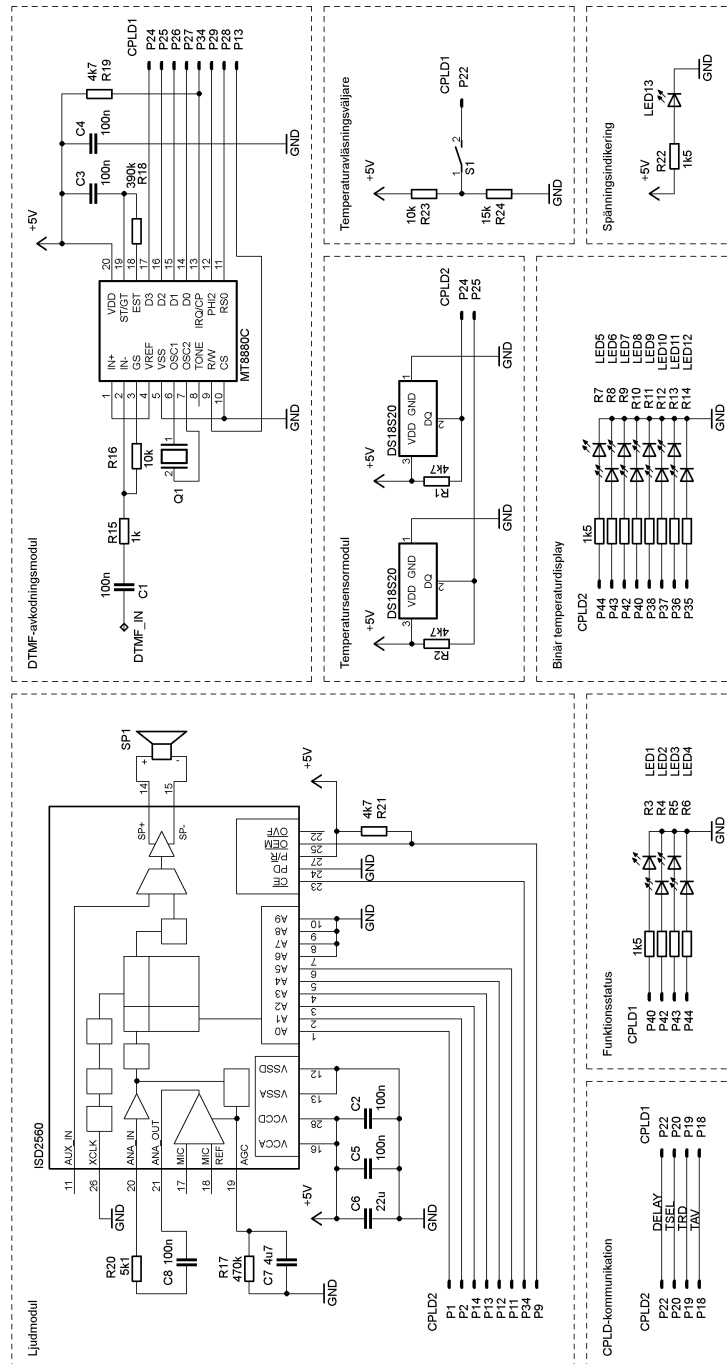
Figur 14: Internt blockschema, moduler (CPLD2)

C Komponentlista

Tabell 3: Komponentlista, tabell

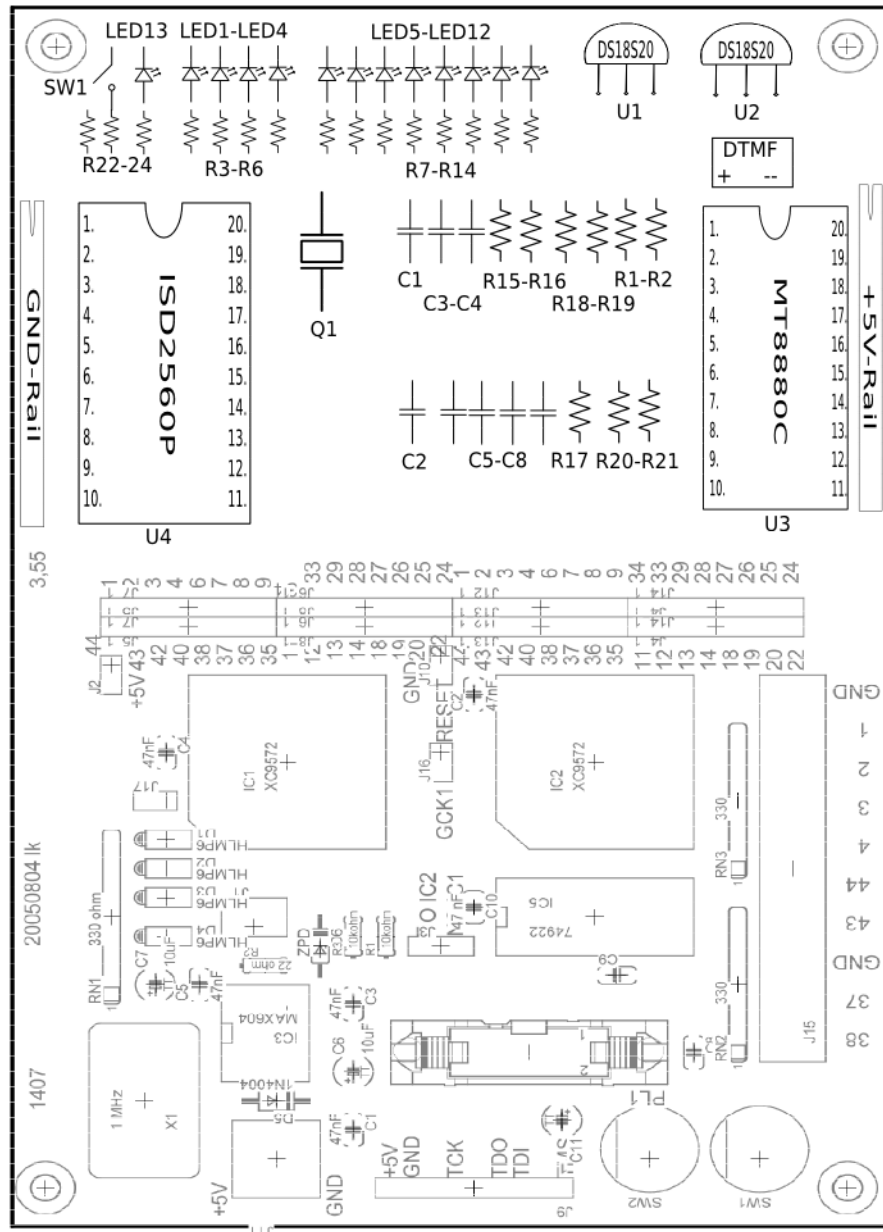
Komponenter			
Part	Value	Device	Manufacturer
C1-C5, C8	100n	capacitor	-
C6	22u	capacitor	-
C7	4u7	capacitor	-
LED1-LED13	-	led	-
Q1	3.579545MHz	crystal	-
R1-R2, R19, R21	4k7	resistor	-
R3-R14, R22	1k5	resistor	-
R15	1k	resistor	-
R16, R23	10k	resistor	-
R17	470k	resistor	-
R18	390k	resistor	-
R20	5k1	resistor	-
R24	15k	resistor	-
S1	-	switch	-
SW1	-	speaker	-
U1	-	DS18S20	Maxim
U2	-	DS18S20	Maxim
U3	-	MT8880C	Mitel
U4	-	ISD2560	Futurlec

D Kretsschema



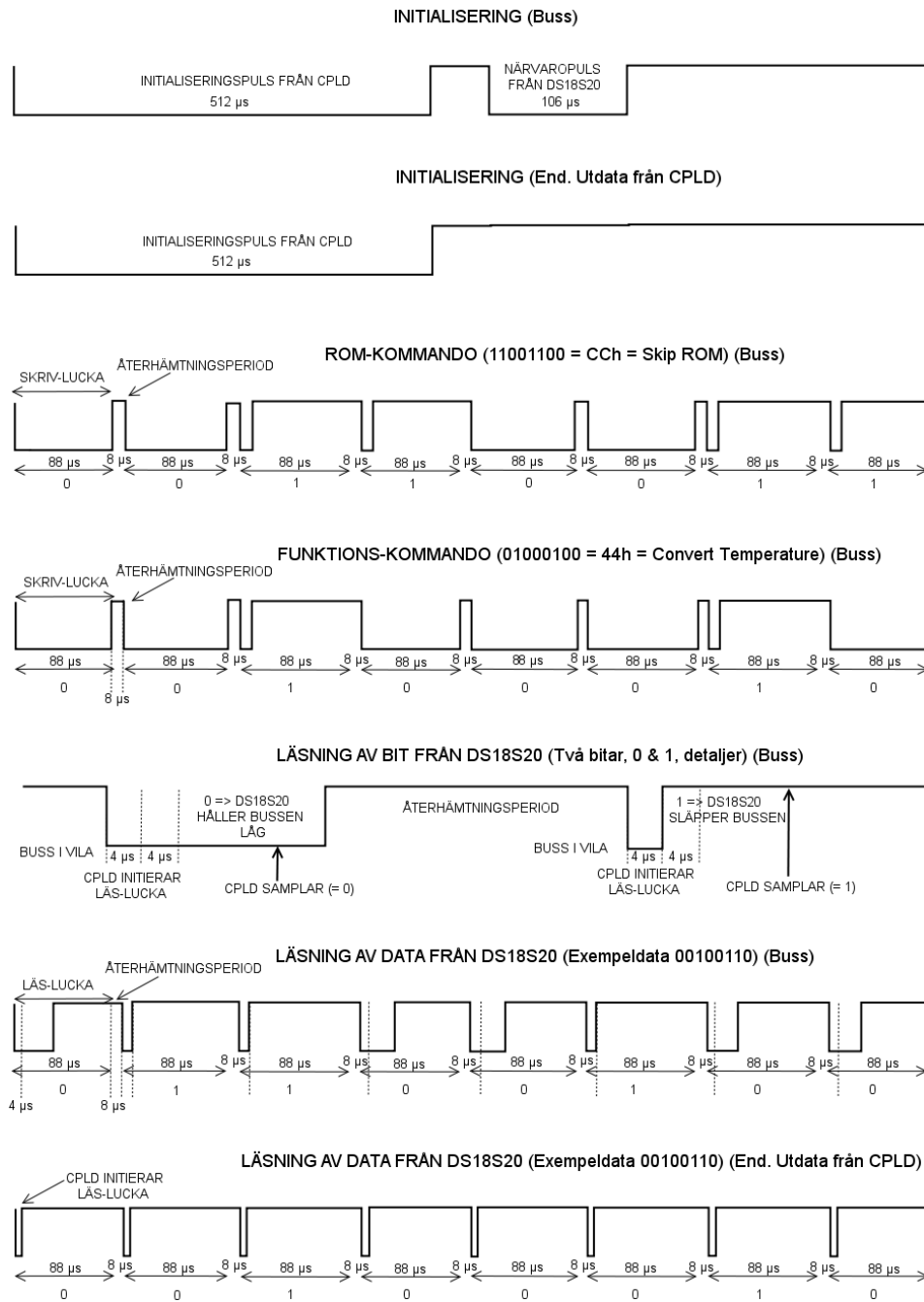
Figur 15: Kretsschema

E Kretskortslayout



Figur 16: Kretskortslayout

F Timingdiagram



Figur 17: Timingdiagram för Temperaturmodulen

G Signallista

Tabell 4: Signallista, tabell

Signaler			
Signal	Från	Till	Beskr.
clk	Extern	Allt	Global Klocksignal
rst	Extern	Allt	Global Resetsignal
buttons[3..0]	Knappsats	Knappsatsmodul	Indatavektor
kbDav	Knappsats	Knappsatsmodul	Availablesignal
funcData[3..0]	Funktionsmodul	FunktionsLEDs	Funktionsstyrning
dtmfData[3..0]	DTMF-modul	MT8880	Databuss till MT8880 (bidir.)
$\Phi 2$	DTMF-modul	MT8880	Klocksignal till MT8880
r/w	DTMF-modul	MT8880	Read/Write till MT8880
rs0	DTMF-modul	MT8880	Intieringssignal till MT8880
irq	MT8880	DTMF-modul	Interruptsignal från MT8880
kData[3..0]	Knappsatsmodul	Styrenhet	Data från knappsatsmodul
kAv	Knappsatsmodul	Styrenhet	Availablesignal från knappsatsmodul
kAck	Styrenhet	Knappsatsmodul	Ack. till knappsatsmodul
tSel	Styrenhet	Temperaturmodul	Sensorselectsignal
tRd	Styrenhet	Temperaturmodul	Read-signal (starta läsning)
tAv	Temperaturmodul	Styrenhet	Availablesignal, valid data
sData[3..0]	Styrenhet	Ljudmodul	Ljudadressbuss
sSel	Styrenhet	Ljudmodul	Selectsignal för temperatur/ljud
sPlay	Styrenhet	Ljudmodul	Play-signal (spela ljud)
sDone	Ljudmodul	Styrenhet	Signalerar uppspelning färdig
dData[3..0]	DTMF-modul	Styrenhet	DTMF-databuss
dAv	DTMF-modul	Styrenhet	Availablesignal från DTMF-modul
dAck	Styrenhet	DTMF-modul	Acknowledgement från Styrenhet
fData[3..0]	Styrenhet	Funktionsmodul	Funktionsstatusbuss
dq0	Temperaturmodul	DS18S20	Seriell 1-trådsbuss (bidir.)
dq1	Temperaturmodul	DS18S20	Seriell 1-trådsbuss (bidir.)
temp[7..0]	Temperaturmodul	Ljudmodul	Temperaturdatabuss
addr[4..0]	Ljudmodul	ISD2560P	Adressbuss till ISD2560P
ce	Ljudmodul	ISD2560P	Chip-Enable från ljudmodul
eom	IDS2560P	Ljudmodul	End-Of-Message från ISD2560P

H Arbetsfördelning

Det har varit mycket samarbete, där alla samarbetat med alla olika moduler. Nedan följer en uppdelning i stora drag vem haft det övergripande ansvaret för de större momenten.

Alla har jobbat med felsökning, rapportskrivning och uppkoppling.

Tabell 5: Arbetsfördelning, tabell

Arbetsfördelning

Namn

Moduler / Ansvar

Övrigt

Fredrik Brosser

Temperaturmodul, DS18S20, Styrenhet, Funktioner/Knappsats

Rapport, Blockschema, Tillståndsdigram

Karl Buchka

Ljudmodul, ISD2560P

Rapport

Andreas Henriksson

DTMF-modul, Ljudmodul, Temperaturmodul, MT8880C, ISD2560P

Rapport, Kretsschema

Johan Wolgers

DTMF-modul, MT8880C

Rapport

I Datablad

Tabell 6: Datablad, tabell

Datablad	
Krets	Datablad
DS18S20	http://datasheets.maxim-ic.com/en/ds/DS18S20.pdf
ISD2560P	http://www.futurlec.com/Others/ISD2560P.shtml
MT8880C	http://www.cse.chalmers.se/edu/year/2010/course/EDA234/datablad/mt8880.pdf

Länkar uppdaterade 2011-12-11

J Programlistningar

TemperatureModule.vhd

```

1  -----
   -- Temperature Module
   -- DS18S20 1-Wire Communication
   -- EDA234, Group 2
   --
6  FILE
   -- tempModule.vhd
   -- Last Updated: 2011-12-11
   --
   -- VERSION
11  -- Hardware ("production") v1.5
   --
   -- HARDWARE
   -- Target Device: XC9572XL
   --
16  -- DESCRIPTION
   -- Temperature module connected to two DS18S20 temperature sensors
   -- communicating via a 1-wire serial protocol.
   -- Module has to be reset, then the control unit can request a (by setting tRd
   --   high)
   -- temperature read/conversion from the selected temperature sensor
21  -- (tSel = 0 or 1 for sensor 0 and 1, respectively). When there is
   -- valid data on the bus (conversion and read cycle finished), the
   -- temperature module responds by setting tAv (temperature Available) high.
   -- The temperature can the be read from the bus (temp[7..0]).
   --
26  -----
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
31
Entity temperatureModule is
  port(
    -- Global clock
    clk      : in    std_logic;
    -- Global reset (Internal)
    rstInt   : in    std_logic;
36    -- Temperature Read, trigger signal from Control Unit
    tRd      : in    std_logic;
    -- Signal to MUX, for selecting active sensor (0/1 for dq0/dq1, resp.)
    tSel     : in    std_logic;
41
    -- Temperature Available, indicates valid data on temperature output bus
    tAv      : out   std_logic;
    -- Internal temperature data output
    temp     : out   std_logic_vector(7 downto 0);
46
    -- Output to 1-Wire bus 1 (temperature sensor 0)
    dq0      : inout std_logic;
    -- Output to 1-Wire bus 0 (temperature sensor 1)
    dq1      : inout std_logic
51  );
end temperatureModule;

Architecture Behavioral of temperatureModule is
56  -- Internal signal declarations
   -- Buffer Enable
   signal E      : std_logic;
   signal nextE  : std_logic;
61
   -- Valid data on temperature bus
   signal tAvInt : std_logic;
   signal nexttAvInt : std_logic;
66
   -- State variable (as integer)
   signal state  : integer range 0 to 15;
   signal nextState : integer range 0 to 15;
71
   -- Data to be sent on bus
   signal data   : std_logic_vector(7 downto 0);
   signal nextData : std_logic_vector(7 downto 0);
76
   -- Internal temperature data output
   signal tempOut : std_logic_vector(7 downto 0);
   signal nexttempOut : std_logic_vector(7 downto 0);

```

```

81  -- Reading sign bit from sensor
    signal signBit      : std_logic;
    signal nextSignBit  : std_logic;

    -- Sampling of bus by master
    signal sample       : std_logic;
    signal nextSample   : std_logic;

86  -- Counter used when sending a logical 0 on bus ('Zero Counter')
    signal ZC           : std_logic_vector(3 downto 0);
    signal nextZC       : std_logic_vector(3 downto 0);

91  -- Signal for keeping track of our progress through the read-cycle
    signal progress     : std_logic_vector(1 downto 0);
    signal nextProgress : std_logic_vector(1 downto 0);

    -- Counter for keeping track of which bit we are currently transmitting or
    -- sampling
    signal bitCnt       : std_logic_vector(2 downto 0);
96  signal nextBitCnt   : std_logic_vector(2 downto 0);

    -- Internal counter used to create timing pulses
    signal cntInt       : std_logic_vector(8 downto 0);
    signal nextCntInt   : std_logic_vector(8 downto 0);

101 -- Timing pulses, 512, 256, 8 and 4 us, respectively
    signal delayLong    : std_logic;
    signal delayMedium  : std_logic;
    signal delayShort   : std_logic;
106 signal delayTiny    : std_logic;

    -- Constants related to timing
    constant LongDelayConstant : std_logic_vector := "111111110";
    constant MediumDelayConstant : std_logic_vector := "11111111";
111 constant ShortDelayConstant : std_logic_vector := "111";
    constant TinyDelayConstant : std_logic_vector := "11";

    -- Base value (reset) for ZC
    constant ZCrst          : std_logic_vector := "1010";

116 -- Begin architecture
    begin

    -- Assign internal temperature available signal to output
121 tAv      <= tAvInt;
    -- Assign internal temperature bus to output
    temp    <= tempOut;

126 -- SyncP, synchronous (clocked) process responsible for clocking in the new
    -- states according to nextState
    --
    -- NB! : This is the only clocked process,
    -- keeping track of all state or value updates (current => next)
131 --
    SyncP : process(clk, rstInt)
    begin
        if(not(rstInt) = '1') then
136 state      <= 0;
            cntInt <= (others => '0');
            progress <= (others => '0');
            bitCnt <= (others => '1');
            data    <= (others => '1');
141 tempOut    <= (others => '1');
            ZC      <= ZCrst;
            sample  <= '1';
            E       <= '0';
            signBit <= '0';
146 tAvInt     <= '0';
        elseif(clk'Event and clk = '1') then
            state <= nextState;
            ZC    <= nextZC;
            E     <= nextE;
151 -- Increment internal counter
            cntInt <= nextCntInt;
            progress <= nextProgress;
            bitCnt <= nextBitCnt;
            data    <= nextData;
156 sample      <= nextSample;
            tempOut <= nexttempOut;
            signBit <= nextSignBit;
            tAvInt  <= nexttAvInt;
        end if;
    end process;

```

```

161  end process;

-- BusP, process responsible for handling the buffered output to the bus,
-- according to the enable signal.
166  -- Works as a buffer and MUX for the 1-wire buses
--

BusP : process(E, tSel)
begin
171  -- Default : both buses in threestate
  dq0 <= 'Z';
  dq1 <= 'Z';
  -- Drive selected bus low if output enabled
  if (E = '1') then
176    if (tSel = '0') then
      dq0 <= '0';
      elsif (tSel = '1') then
181        dq1 <= '0';
      end if;
    end if;
  end process;

-- CountP, internal counter responsible for creating pulses with certain
186  -- time intervals. Uses a local (to the Architecture) counter variable.
--

CountP : process(cntInt)
begin
191  -- Increment internal counter
  nextCntInt <= cntInt + 1;
  -- Gives pulses every 4 us
  if (cntInt(1 downto 0) = TinyDelayConstant) then
196    delayTiny <= '1';
  else
    delayTiny <= '0';
  end if;
  -- Gives pulses every 8 us
  if (cntInt(2 downto 0) = ShortDelayConstant) then
201    delayShort <= '1';
  else
    delayShort <= '0';
  end if;
  -- Gives pulses every 256 us
206  if (cntInt(7 downto 0) = MediumDelayConstant) then
    delayMedium <= '1';
  else
    delayMedium <= '0';
  end if;
  -- Gives pulses every 512 us
211  if (cntInt = LongDelayConstant) then
    delayLong <= '1';
    nextCntInt <= (others => '0');
  else
216    delayLong <= '0';
  end if;
end process;

-- ComP, State Machine handling the master side of the 1-wire bus
221  -- communication with the DS18S20. Divided into stages/modes as follows:
--
-- 1. INIT (Reset - Presence pulses)
-- 2. SEND (Transmission of data from Master to DS18S20)
226  -- 3. READ (Master reads data from DS18S20)
-- 4. IDLE (Bus is idle, pulled high by pull-up resistor)
--

ComP : process(tRd, state, delayLong, delayMedium, delayShort, delayTiny,
  progress, ZC, bitCnt, tSel, dq0, dq1, sample, data, E, tempOut, signBit,
  tAvInt)
231  begin
  -- Defaults
  nextState <= state;
  nextProgress <= progress;
236  nextBitCnt <= bitCnt;
  nextZC <= ZC;
  nextSample <= sample;
  nextData <= data;
  nextE <= E;
241  nexttempOut <= tempOut;
  nextSignBit <= signBit;

```

```

nextAvInt    <= tAvInt;

case state is
246
-- INIT
when 0 =>
251   -- Initialization of signals
   nextProgress <= "00";
   nextZC <= ZCrst;
   nextSignBit <= '0';
   nextE <= '0';
256   -- Wait for trigger from control unit
   if(tRd = '1') then
      nextState <= state + 1;
      -- Entering the read cycle - data no longer valid
      nextAvInt <= '0';
261   end if;
when 1 =>
   -- Enable output and send logical 0
   nextE <= '0';
   if(delayLong = '1') then
266     nextState <= state + 1;
   end if;
when 2 =>
   nextE <= '1';
   if(delayLong = '1') then
271     nextState <= state + 1;
     -- CCh, Skip ROM
     nextData <= X"CC";
   end if;
276   when 3 =>
     -- Put bus into threestate and wait for response
     nextE <= '0';
     if(delayLong = '1') then
        nextState <= state + 1;
     end if;
281
-- SEND
when 4 =>
286   -- Prepare for transmit of the byte in data
   -- Release bus and delay
   nextE <= '0';
   if(delayShort = '1') then
      nextState <= 5;
291   end if;
   -- Send logical 0 or 1 by driving bus low for a certain number of
   -- shortDelay periods (1 for 1's, ZCrst for 0's)
when 5 =>
   -- Send logical 1
   if(data(7-conv_integer(bitCnt)) = '1' and delayShort = '1') then
296     nextE <= '0';
     -- Write slot complete, reset iterator and send next bit
     if(ZC = "0000") then
        nextState <= state + 1;
        nextZC <= ZCrst;
301     -- Drive bus low for one delay period
     elsif(ZC = ZCrst) then
        nextE <= '1';
        NextZC <= (ZC - 1);
        -- Decrement iterator, bus is pulled high
306     else
        NextZC <= (ZC - 1);
     end if;
   -- Send logical 0
   elsif(data(7-conv_integer(bitCnt)) = '0' and delayShort = '1') then
311     nextE <= '1';
     -- Write slot complete, reset iterator and send next bit
     if(ZC = "0000") then
        nextState <= state + 1;
        nextZC <= ZCrst;
316     -- Decrement iterator, bus is kept low
     else
        NextZC <= (ZC - 1);
     end if;
   end if;
321   -- Recovery time between transmitted bits, and decrementing bit counter
when 6 =>
   nextE <= '0';
   if(delayShort = '1') then
      if(bitCnt = "000") then

```

```

326      -- Done sending byte, move on and reset bit counter
      nextState <= state + 1;
      nextBitCnt <= (others => '1');
      else
331      -- Send next bit
      nextBitCnt <= bitCnt - 1;
      nextState <= state - 1;
      end if;
    end if;
    -- Done sending Command, disable buffer and prepare to send next byte,
    -- or if the temp sensor is converting temperature, wait for it to finish
336  when 7 =>
    nextE <= '0';
    if(delayShort = '1') then
      -- Converting temperature, go to read
341      if(progress = "01") then
        nextState <= 9;
        -- Move on
      else
        nextState <= state + 1;
346      end if;
    end if;
    -- Prepare to send next Command or start reading temperature
    when 8 =>
    nextE <= '0';
    -- Increase progress variable. NB: assignment at end of process, will
    -- compare with 'old' value!
    nextProgress <= progress + 1;
    case progress is
      when "00" =>
        -- Issue Convert T Command (44h)
356      nextData <= X"44";
        nextState <= 4;
      when "01" =>
        -- Do reset and Skip ROM (CCh)
361      nextData <= X"CC";
        nextState <= 1;
      when "10" =>
        -- Issue Read Scratchpad Command (BEh)
        nextData <= X"BE";
        nextState <= 4;
366      when "11" =>
        -- Master goes into Rx mode
        nextState <= state + 1;
        nextProgress <= "11";
      when others =>
371      -- We should not be here, something is terribly wrong: do full
        -- reset and start over
        nextProgress <= "00";
        nextState <= 0;
    end case;
376
-- READ
--
--
381 -- Master reads 9 bytes from the bus, starting with LSB of Byte 0
-- However, we are only interested in the temperature registers (Byte 0 and 1),
-- so a reset pulse is given after nine bits (8 temp + 1 sign) have been read,
-- telling the DS18S20 to discontinue transfer.
--
386
-- Delay and prepare for read phase
when 9 =>
  nextE <= '0';
  if(delayMedium = '1') then
391    nextState <= state + 1;
  end if;
  -- Pull bus low and wait for response (initiate read time slot, Tinit = 4
  -- us)
  when 10 =>
    nextE <= '1';
    if(delayTiny = '1') then
396      nextState <= state + 1;
    end if;
    -- Release bus and allow pullup resistor to perform its magic (Trc = 4 us)
  when 11 =>
    nextE <= '0';
    if(delayTiny = '1') then
401      nextState <= state + 1;
    end if;
    -- Sample bus (Tsample = 4 us)
  when 12 =>
406

```

```

nextE <= '0';
-- MUX'ed sampling from buses
if(tSel = '0') then
nextSample <= dq0;
411 elsif(tSel = '1') then
nextSample <= dq1;
end if;
if(delayTiny = '1') then
nextState <= state + 1;
416 end if;
-- Recovery time between read slots
when 13 =>
nextE <= '0';
if(delayMedium = '1') then
421 nextState <= state + 1;
end if;
-- Go back and sample next bit (or wait for conversion to finish)
when 14 =>
nextE <= '0';
426 -- Reading 9th bit (temperature sign) and finishing up reading
if(signBit = '1') then

-- Negative temperature, perform 2-com. to sign-value conv.
if(sample = '1') then
431 nexttempOut <= (not(tempOut)) + 1;
end if;

nexttempOut(7) <= sample;
nextState <= state + 1;
436 nextProgress <= "00";
else
-- Waiting for conversion to finish
if(progress = "01") then
if(sample = '0') then
441 nextState <= 10;
elsif(sample = '1') then
nextState <= 8;
else
nextState <= 10;
446 end if;
elsif(progress = "11") then
nextState <= 10;
-- Reading temperature bit
if(bitCnt = "000") then
451 nextBitCnt <= (others => '1');
nexttempOut(7-conv_integer(bitCnt)) <= sample;
nextSignBit <= '1';
else
nextBitCnt <= bitCnt - 1;
456 nexttempOut(7-conv_integer(bitCnt)) <= sample;
end if;
-- Should not be here. If we are, go back and read again
else
nextState <= 10;
461 end if;
end if;

when 15 =>
-- Data on TOut bus is now valid. Go back and wait for next trigger.
466 nexttAvInt <= '1';
nextE <= '0';
nextState <= 0;

-- Other states. Should never be here.
471 when others =>
nextE <= '0';
nextState <= 0;

476 end case;
end process;
end Behavioral;

```

ControlUnit.vhd

```

2 -- Control Unit
-- EDA234, Group 2
--
-- FILE
-- ControlUnit.vhd
7 -- Last Updated: 2011-12-11
--

```

```

-- VERSION
-- Hardware ("production") v1.1
--
12 -- HARDWARE
-- Target Device: XC9572XL
--
-- DESCRIPTION
-- The Control Unit. Responsible for the complete system usage cycle.
17 --
-----

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

Entity ControlUnit is
  port(
27     -- Global clock
        clk      : in    std_logic;
        -- Global reset
        rstInt    : in    std_logic;

32     -- Switch for ON/OFF (Human Interface)
        button    : in    std_logic;

        -- TEMPERATURE MODULE
        -- Read Temperature command signal to Temperature module
        tRd       : out   std_logic;
37     -- Select signal for temperature sensors (sensor 0/1)
        tSel      : out   std_logic;
        -- Temperature Available signal from Temperature module
        tAv       : in    std_logic;

42     -- FUNCTION MODULE
        -- Function control output to Function module
        func      : out   std_logic_vector(3 downto 0);
        -- Enable signal for Function module
        en        : out   std_logic;

47     -- DTMF MODULE
        -- Data input from DTMF module
        dData     : in    std_logic_vector(3 downto 0);
        -- Data available signal from DTMF module
52     dAv        : in    std_logic;
        -- Acknowledgement signal to DTMF module
        dAck      : out   std_logic;

        -- Call In Progress-signal
57     cIP        : in    std_logic;

        -- KEYBOARD MODULE
        -- Data input from keyboard
        kData     : in    std_logic_vector(3 downto 0);
62     -- Data available signal from keyboard
        kAv       : in    std_logic;
        -- Acknowledgement signal from keyboard
        kAck      : out   std_logic;
        -- Keyboard Output Enable
67     kOe        : out   std_logic;

        -- SOUND MODULE
        -- Done (playing sound) signal from Sound module
72     sDone      : in    std_logic;
        -- Play signal to Sound module
        sPlay     : out   std_logic;
        -- Select signal to Sound module (temp / sound)
        sSel      : out   std_logic;
        -- Address bus to Sound Module
77     sAddr      : out   std_logic_vector(3 downto 0)

    );
end ControlUnit;
82

Architecture Behavioral of ControlUnit is

    -- State variable (as integer)
    signal state      : integer range 0 to 3;
87    signal nextState : integer range 0 to 3;

    signal funcStatus : std_logic_vector(3 downto 0);
    signal nextFuncStatus : std_logic_vector(3 downto 0);

92    signal tSelStatus : std_logic;

```



```

97  signal nextTSelStatus    : std_logic;

    signal sAddrInt        : std_logic_vector(3 downto 0);
    signal nextSAddrInt    : std_logic_vector(3 downto 0);

102  signal sPlayInt        : std_logic;
    signal nextSPlayInt    : std_logic;

    signal sSelInt         : std_logic;
    signal nextSSelInt     : std_logic;

    begin

        tSel <= tSelStatus;
107    func <= funcStatus;
        sAddr <= sAddrInt;
        sPlay <= sPlayInt;
        sSel <= sSelInt;

112    -- Synchronous (clocked) process, handing state changes
    syncP : process(clk, rstInt)
    begin
        if(not(rstInt) = '1') then
117            state <= 0;
            tSelStatus <= '0';
            funcStatus <= (others => '0');
            sAddrInt <= (others => '0');
            sPlayInt <= '0';
            sSelInt <= '0';
122        elsif(clk'Event and clk = '1') then
            state <= nextState;
            tSelStatus <= nextTSelStatus;
            funcStatus <= nextFuncStatus;
            sAddrInt <= nextSAddrInt;
127            sPlayInt <= nextSPlayInt;
            sSelInt <= nextSSelInt;
        end if;
    end process;

132    commandP : process(dAv, dData, kAv, kData, funcStatus, tSelStatus, sSelInt,
        sAddrInt, sPlayInt)
    begin

        kOe <= '0';
        nextTSelStatus <= TSelStatus;
137    nextFuncStatus <= FuncStatus;
        nextSAddrInt <= SAddrInt;
        nextSSelInt <= SSelInt;
        nextSPlayInt <= SPlayInt;

142    dAck <= '0';
        kAck <= '0';
        en <= '0';

147    -- Command from Manual Control Panel
    if(kAv = '1') then
        kAck <= '1';
        case kData(2 downto 0) is
            when "000" => -- 1
152                nextSAddrInt <= "1111";
                nextSSelInt <= '1';
                nextSPlayInt <= '1';
                en <= '1';
                nextFuncStatus(0) <= '1';
            when "001" => -- 2
157                en <= '1';
                nextFuncStatus(1) <= '1';
            when "010" => -- 3
                en <= '1';
                nextFuncStatus(2) <= '1';
162            when "011" => -- A
                nextTSelStatus <= '0';
            when "100" => -- 4
                en <= '1';
                nextFuncStatus(0) <= '0';
167            when "101" => -- 5
                en <= '1';
                nextFuncStatus(1) <= '0';
            when "110" => -- 6
                en <= '1';
                nextFuncStatus(2) <= '0';
172            when "111" => -- B
                nextTSelStatus <= '1';
            when others =>

```

```

177         end case;
        end if;

    end process;

182    -- Process handling the temperature reading and sounds
    tempP : process(button, state, tAv)
    begin
        -- Defaults
        nextState <= state;

187        case state is
            when 0 =>
                -- OFF State
                tRd <= '0';
                if(button = '1') then
192                    nextState <= state + 1;
                end if;
            when 1 =>
                tRd <= '1';
                nextState <= state + 1;
197            when 2 =>
                tRd <= '0';
                nextState <= state + 1;
            when 3 =>
                tRd <= '0';
                if(tAv = '1') then
202                    nextState <= 0;
                end if;
            end case;

207        case state is
            when 0 =>
                -- Wait for call in progress
                if(cIP) then
212                    nextState <= state + 1;
                    tRd <= '1';
                    nextTSelStatus <= '0';
                    sData <= "0000";
                    sPlay <= '1';
                    sSel <= '1';
                end if;
            when 1 =>
                -- Play sound ("temperaturen inne ar")
                if(sDone = '1' and tAv = '1') then
222                    sSel <= '0';
                    tRd <= '0';
                    sPlay <= '1';
                    nextState <= state + 1;
                end if;
            when 2 =>
                -- Play indoor temperature
227                if(sDone = '1') then
                    sSel <= '1';
                    tRd <= '1';
                    nextTSelStatus <= '1';
                    sData <= "0001";
232                    sPlay <= '1';
                    nextState <= state + 1;
                end if;
            when 3 =>
                -- Play sound ("temperaturen ute ar")
237                if(sDone = '1' and tAv = '1') then
                    sSel <= '0';
                    tRd <= '0';
                    sPlay <= '1';
                    nextState <= state + 1;
242                end if;
            when 4 =>
                -- Play outdoor temperature
                if(sDone = '1') then
247                    sSel <= '1';
                    sData <= "0100";
                    sPlay <= '1';
                    nextState <= state + 1;
                end if;
            when 5 =>
                -- Play sound ("Funktionsstyrning")
252                if(sDone = '1') then
                    nextState <= state + 1;
                end if;
            when 6 =>
                -- Wait for input
257                if(cIP = '0') then

```

```

-- Call over, go back to idle state
nextState <= 0;
262 elseif(dAv = '1') then
    dAck <= '1';
    sSel <= '1';
    -- Respond to input
    case dData is
267     when "0001" => -- 1
        en <= '1';
        nextFuncStatus(0) <= '1';
        sData <= "0101"; -- F1
        sPlay <= '1';
272     nextState <= 7;
    when "0010" => -- 2
        en <= '1';
        nextFuncStatus(1) <= '1';
        sData <= "0110"; -- F2
277     sPlay <= '1';
        nextState <= 8;
    when "0011" => -- 3
        en <= '1';
        nextFuncStatus(2) <= '1';
282     sData <= "0111"; -- F3
        sPlay <= '1';
        nextState <= 9;
    when "0100" => -- 4
        en <= '1';
        nextFuncStatus(0) <= '0';
287     sData <= "0101"; -- F1
        sPlay <= '1';
        nextState <= 7;
    when "0101" => -- 5
        en <= '1';
        nextFuncStatus(1) <= '0';
292     sData <= "0110"; -- F2
        sPlay <= '1';
        nextState <= 8;
    when "0110" => -- 6
        en <= '1';
        nextFuncStatus(2) <= '0';
297     sData <= "0111"; -- F3
        sPlay <= '1';
        nextState <= 9;
    when "0111" => -- 7
        en <= '1';
        nextFuncStatus(3) <= '1';
302     sData <= "1000"; -- F4
        sPlay <= '1';
    when "1000" => -- 8
        en <= '1';
        nextFuncStatus(3) <= '0';
307     sData <= "1000"; -- F4
        sPlay <= '1';
        nextState <= 10;
    when "1001" => -- 9
        nextTSelStatus <= '1';
312     when "1010" => -- 0
        nextTSelStatus <= '0';
    when others =>
        end case;
    end if;
322 when 7 =>
    -- Play sound ("funktion 1")
    if(sDone = '1') then
        sSel <= '1';
        sPlay <= '1';
327     nextState <= 11;
        if(funcStatus(0) = '1') then
            sData <= "1001"
        else
            sData <= "1010"
        end if;
    end if;
332 when 8 =>
    -- Play sound ("funktion 2")
    if(sDone = '1') then
        sSel <= '1';
        sPlay <= '1';
337     nextState <= 11;
        if(funcStatus(1) = '1') then
            sData <= "1001"
        else
            sData <= "1010"
342

```

```

347         end if;
        end if;
        when 9 =>
            -- Play sound ("funktion 3")
            if(sDone = '1') then
                sSel <= '1';
                sPlay <= '1';
                nextState <= 11;
352             if(funcStatus(2) = '1') then
                 sData <= "1001"
            else
                sData <= "1010"
            end if;
357         end if;
        when 10 =>
            -- Play sound ("funktion 4")
            if(sDone = '1') then
                sSel <= '1';
                sPlay <= '1';
362             nextState <= 11;
            if(funcStatus(3) = '1') then
                sData <= "1001"
            else
                sData <= "1010"
            end if;
367         end if;
        when 11 =>
            -- Play sound ("Till/Fran")
372             if(sDone = '1') then
                -- Go back to idle state and wait for next command
                nextState <= 6;
            end if;
377     end process;
end Behavioral;

```

DTMFModule.vhd

```

-- DTMF Module
-- EDA234, Group 2
--
5  -- FILE
-- DTMFModule.vhd
-- Last Updated: 2011-12-09
--
-- VERSION
10 -- Hardware ("production") v1.1
--
-- HARDWARE
-- Target Device: XC9572XL
--
15 -- DESCRIPTION
-- DTMF Module, responsible for decoding DTMF data and presenting it
-- to the control unit.
--
20 library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity DTMFModule is
25 port(
    -- Clock
    clk          : in std_logic;
    -- Asynchronous reset
    rst          : in std_logic;
    -- Start signal
30 sigInit      : in std_logic;
    -- Early steering from DTMF chip via IRQ pin
    irq          : in std_logic;
    -- Acknowledge signal from control unit
    ack          : in std_logic;
35 -- Input vector
    extDataBus   : inout std_logic_vector (3 downto 0);
    -- Phi2 clock
    phi2         : out std_logic;
    -- Read/Write select
40 rw           : out std_logic;
    -- Register select
    rs0          : out std_logic;
    -- Data valid signal to control unit
    dav          : out std_logic;

```

```

45  -- Data bus to control unit
    intDataBus      : out std_logic_vector (3 downto 0)
    );
end DTMFModule;
50
architecture Behavioral of DTMFModule is
    -- State declaration
    type stateType is ( s0,  s1,  s2,  s3,  s4,  s5,  s6,  s7,  s8,  s9,  s10, s11,
                        s12, s13, s14, s15,
55      s16, s17, s18, s19, s20, s21, s22, s23, s24, s25, s26, s27, s28, s29)
    ;
    -- Current state
    signal state      : stateType;
    -- Next state
    signal nextState  : stateType;
60  -- Data to control unit
    signal dataToControl : std_logic_vector(3 downto 0);
    -- Next data to control unit
    signal nextDataToControl : std_logic_vector(3 downto 0);
    -- Output drive enable
65  signal outputEnable : std_logic;
    -- Internal output data bus
    signal outDataBus   : std_logic_vector(3 downto 0);
    -- Internal input data bus

70 begin
    -- Synchronous process controlling state changes
    syncP : process(clk, rst)
    begin
        -- Asynchronous reset, active high
75      if (rst = '0') then
          state <= s0;
          dataToControl <= "0000";
        -- Trigger state changes on positive clock flank
        elsif (clk'Event and clk = '1') then
80          state <= nextState;
          dataToControl <= nextDataToControl;
        end if;
    end process;

    -- Asynchronous process describing states
85  stateP : process(state, sigInit, dataToControl, irq, extDataBus, ack)
    begin
        -- Assign default values to signals
        phi2 <= '0';
        rw <= '0';
90      rs0 <= '0';
        outputEnable <= '0';
        outDataBus <= "0000";
        --inDataBus <= "0000";
95      nextState <= state;
        dav <= '0';
        nextDataToControl <= dataToControl;

        -- Send data to control unit
100     intDataBus <= dataToControl;

        case state is
            when s0 =>
                if (sigInit = '0') then
105                 nextState <= s1;
                end if;

            -- 1) Read Status Register
            when s1 =>
110                 rs0 <= '1';
                 rw <= '1';
                 nextState <= s2;
            when s2 =>
115                 rs0 <= '1';
                 rw <= '1';
                 phi2 <= '1';
                 nextState <= s3;
            when s3 =>
120                 rs0 <= '1';
                 rw <= '1';
                 nextState <= s4;

            -- 2) Write "0000" to Control Register A
            when s4 =>
125                 rs0 <= '1';
                 outputEnable <= '1';

```

```

130     nextState <= s5;
    when s5 =>
        rs0 <= '1';
        phi2 <= '1';
        outputEnable <= '1';
        nextState <= s6;
    when s6 =>
135         rs0 <= '1';
        outputEnable <= '1';
        nextState <= s7;

    -- 3) Write "0000" to Control Register A
    when s7 =>
140         rs0 <= '1';
        phi2 <= '1';
        outputEnable <= '1';
        nextState <= s8;
    when s8 =>
145         rs0 <= '1';
        outputEnable <= '1';
        nextState <= s9;

    -- 4) Write "1000" to Control Register A
    when s9 =>
150         rs0 <= '1';
        outDataBus <= "1000";
        outputEnable <= '1';
        nextState <= s10;
    when s10 =>
155         rs0 <= '1';
        phi2 <= '1';
        outDataBus <= "1000";
        outputEnable <= '1';
        nextState <= s11;
    when s11 =>
160         rs0 <= '1';
        outDataBus <= "1000";
        outputEnable <= '1';
        nextState <= s12;
165         nextState <= s12;

    -- 5) Write "0000" to Control Register B
    when s12 =>
170         rs0 <= '1';
        outputEnable <= '1';
        nextState <= s13;
    when s13 =>
175         rs0 <= '1';
        phi2 <= '1';
        outputEnable <= '1';
        nextState <= s14;
    when s14 =>
180         rs0 <= '1';
        outputEnable <= '1';
        nextState <= s15;

    -- 6) Read Status Register
    when s15 =>
185         rs0 <= '1';
        rw <= '1';
        nextState <= s16;
    when s16 =>
190         rs0 <= '1';
        rw <= '1';
        phi2 <= '1';
        nextState <= s17;
    when s17 =>
195         rs0 <= '1';
        rw <= '1';
        nextState <= s18;

    -- Chip init ready, now init chip interface
    -- Write "1101" to Control Register A
    -- b0 set Enable tone output
    -- b1 clr Enable DTMF
    -- b2 set Enable IRQ
    -- b3 set Write to CRB in next write phase
    when s18 =>
200         rs0 <= '1';
        outDataBus <= "1101";
        outputEnable <= '1';
        nextState <= s19;
    when s19 =>
205         rs0 <= '1';
        phi2 <= '1';
210

```

```

215         outDataBus <= "1101";
            outputEnable <= '1';
            nextState <= s20;
        when s20 =>
            rs0 <= '1';
            outDataBus <= "1101";
            outputEnable <= '1';
            nextState <= s21;

220         -- Write "0010" to Control Register B
        -- b0 clr Enable burst mode
        -- b1 set Enable test mode
        -- b2 clr Disable single tone generation
        -- b3 clr Don't care as single tone generations is not active
225         when s21 =>
            rs0 <= '1';
            outDataBus <= "0010";
            outputEnable <= '1';
            nextState <= s22;
230         when s22 =>
            rs0 <= '1';
            phi2 <= '1';
            outDataBus <= "0010";
            outputEnable <= '1';
235             nextState <= s23;
        when s23 =>
            rs0 <= '1';
            outDataBus <= "0010";
            outputEnable <= '1';
240             nextState <= s24;

        -- Idle state
        when s24 =>
            -- Wait for DTMF data present in the DTMF chip
245             if (irq = '0') then
                nextState <= s25;
            end if;

        -- Read DTMF data from MT8880C
250         when s25 =>
            rw <= '1';
            nextState <= s26;
        when s26 =>
            rw <= '1';
            phi2 <= '1';
255             nextState <= s27;
        when s27 =>
            rw <= '1';
            phi2 <= '1';
            nextDataToControl <= extDataBus;
260             nextState <= s28;
        when s28 =>
            rw <= '1';
            dav <= '1';
265             nextState <= s29;

        -- Wait for control unit to acknowledge data
        when s29 =>
            dav <= '1';
270             if (ack = '1') then
                nextState <= s24;
            end if;

        when others =>
275             nextState <= s0;

    end case;

end process;

280 -- Asynchronous process controlling tristate outputs
ouputEnableP : process(outputEnable, outDataBus)
begin
    if (outputEnable = '1') then
285         extDataBus <= outDataBus;
    else
        extDataBus <= "ZZZZ";
    end if;
end process;

290 end Behavioral;

```

FunctionModule.vhd

```

-- Function Module
-- EDA234, Group 2
4
-- FILE
-- FunctionModule.vhd
-- Last Updated: 2011-12-08
--
9
-- VERSION
-- Hardware (*production*) v1.3
--
-- HARDWARE
-- Target Device: XC9572XL
14
-- DESCRIPTION
-- The Function Module is a simple module responsible for keeping track of and
-- updating the status (on/off) of the functions used.
--
19
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
24 use IEEE.STD_LOGIC_ARITH.ALL;

Entity FunctionModule is
    port( -- Global clock
29         clk      : in    std_logic;
        -- Global reset
        rstInt    : in    std_logic;
        -- Enable signal from control unit
        en        : in    std_logic;
34         funcIn   : in    std_logic_vector(3 downto 0);
        -- Output to actual functions to be controlled
        funcOut   : out   std_logic_vector(3 downto 0)
    );
end FunctionModule;
39

Architecture Behavioral of FunctionModule is
begin
    funcP : process(funcIn, rstInt, clk, en)
    begin
44         if(not(rstInt) = '1') then
            funcOut <= (others => '0');
            elsif (clk'Event and clk = '1' and en = '1') then
                funcOut <= funcIn;
            end if;
49         end process;
    end Behavioral;

```

KeyboardModule.vhd

```

-- Keyboard Module
-- EDA234, Group 2
4
-- FILE
-- KeyboardModule.vhd
-- Last Updated: 2011-12-06
--
9
-- VERSION
-- Hardware (*production*) v1.2
--
-- HARDWARE
-- Target Device: XC9572XL
14
-- DESCRIPTION
-- Synchronization and Available / Ack Data Transfer from Keyboard
--
19
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
24 use IEEE.STD_LOGIC_ARITH.ALL;

Entity KeyboardModule is
    port( -- Global clock
        clk      : in    std_logic;

```



```

29      -- Global reset
      rstInt      : in      std_logic;
      -- Data Available signal from Keyboard
      kbAvIn      : in      std_logic;
      -- Data from Keyboard
      kbDataIn     : in      std_logic_vector(3 downto 0);
34      -- Acknowledgement signal from Control Unit
      kbAck        : in      std_logic;
      -- Data output to Control Unit
      kbDataOut    : out     std_logic_vector(3 downto 0);
      -- Data Available signal output to Control Unit
39      kbAvOut     : out     std_logic
    );
end KeyboardModule;

Architecture Behavioral of KeyboardModule is
44
    -- State variable (as integer)
    signal state      : integer range 0 to 3;
    signal nextState  : integer range 0 to 3;
    signal kbDataOutInt : std_logic_vector(3 downto 0);
49    signal nextKBDataOutInt : std_logic_vector(3 downto 0);

begin

    -- Concurrent Assignment
54    kbDataOut    <= kbDataOutInt;

    syncP : process(clk, rstInt)
    begin
        if(not(rstInt) = '1') then
59            state <= 0;
            kbDataOutInt <= (others => '0');
        elsif(clk'Event and clk = '1') then
            state <= nextState;
            kbDataOutInt <= nextKBDataOutInt;
64        end if;
    end process;

    keyboardP : process(kbDataIn, kbAvIn, kbAck, state, kbDataOutInt)
69    begin

        -- Defaults
        nextState <= state;
        nextKBDataOutInt <= kbDataOutInt;

74        case state is
            when 0 =>
                kbAvOut <= '0';
                if(kbAvIn = '1') then
                    nextState <= state + 1;
79                end if;
            when 1 =>
                kbAvOut <= '1';
                nextKBDataOutInt <= kbDataIn;
                nextState <= state + 1;
84            when 2 =>
                kbAvOut <= '1';
                if(kbAck = '1') then
                    nextState <= state + 1;
            end if;
89            when 3 =>
                kbAvOut <= '0';
                if(kbAvIn = '0') then
                    nextState <= 0;
            end if;
94        end case;
    end process;

end Behavioral;

```

SoundModule.vhd

```

-- Sound Module
3 -- EDA234, Group 2
--
-- FILE
-- SoundModule.vhd
-- Last Updated: 2011-12-11
8 --
-- VERSION

```

```

-- Hardware ("production") v1.0
--
-- HARDWARE
13 -- Target Device: XC9572XL
--
-- DESCRIPTION
-- Controlling external sound chip
--
18 -----
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

23 entity isdctrl is
    Port ( clk : in STD_LOGIC;
          reset : in STD_LOGIC;

28         -- To/From control unit
          play : in STD_LOGIC;
          ct : in STD_LOGIC;
          ctrlAddr : in STD_LOGIC_VECTOR (3 downto 0);
          done : out STD_LOGIC;

33         -- From temp module
          tempAddr : in STD_LOGIC_VECTOR (7 downto 0);

          -- To/From ISD2560
          eom : in STD_LOGIC;
          outAddr : out STD_LOGIC_VECTOR (5 downto 0);
          ce : out STD_LOGIC);
end isdctrl;

43 architecture Behavioral of isdctrl is
    type state_type is (s0, slc, slt, slto, s2c, s2t, s2to, s3c, s3tm, s3tp, s4tm,
                        s4tp);
    signal state, nstate: state_type;
    begin

    process (eom, play, ct, tempAddr, ctrlAddr, state)
    48 begin
        outAddr <= "000000";
        ce <= '1'; -- Chip Enable active low
        done <= '0';
        nstate <= state;

    53 case state is
        -- Base state
        when s0 =>
            if play = '1' then
            58 if ct = '1' then
                nstate <= slc; -- Control unit address
            else
                if tempAddr(6) = '0' then
                    nstate <= slt; -- Temp unit address
                63 else
                    nstate <= slto; -- Temp overflow
                    end if;
                end if;
            end if;

        68 -- Control unit addressing
        when slc =>
            outAddr <= "10" & ctrlAddr;
            nstate <= s2c;

        73 when s2c =>
            outAddr <= "10" & ctrlAddr;
            ce <= '0';
            if eom = '1' then
                nstate <= s2c; -- Wait for sound to finish
            78 else
                nstate <= s3c;
                end if;
            when s3c =>
                done <= '1';
                nstate <= s0;

            83 -- Temp module addressing
            when slt => -- "Regular" address
                outAddr <= tempAddr(5 downto 0);
                nstate <= s2t;

            88 when slto => -- Overflow case
                outAddr <= "111111";
                nstate <= s2to;
            when s2t =>

```

```

93      outAddr <= tempAddr(5 downto 0);
      ce <= '0';
      if eom = '1' then
          nstate <= s2t; -- Wait for sound to finish
      else
98          if tempAddr(7) = '0' then
              nstate <= s3tm;
              else
                  nstate <= s3tp;
              end if;
103      end if;
      when s2to =>
          outAddr <= "111111";
          ce <= '0';
          if eom = '1' then
108              nstate <= s2to; -- Wait for sound to finish
          else
              if tempAddr(7) = '0' then
                  nstate <= s3tm;
              else
                  nstate <= s3tp;
              end if;
113      end if;
      when s3tm =>
          outAddr <= "100011";
          nstate <= s4tm;
118      when s3tp =>
          outAddr <= "100010";
          nstate <= s4tp;
      when s4tm =>
          outAddr <= "100011";
          ce <= '0';
          if eom = '1' then
123              nstate <= s4tm; -- Wait for sound to finish
          else
              nstate <= s3c;
          end if;
128      when s4tp =>
          outAddr <= "100010";
          ce <= '0';
          if eom = '1' then
133              nstate <= s4tp; -- Wait for sound to finish
          else
              nstate <= s3c;
          end if;
138      when others =>
          end case;
      end process;

      process (clk)
143      begin
          if clk'event and clk = '1' then
              if(not(reset = '1')) then
                  state <= s0;
              else
148                  state <= nstate;
              end if;
          end if;
      end process;

153 end Behavioral;

```