LUNDS UNIVERSITET
Lunds Tekniska Högskola

MIDI-styrd tongenerator med FM-syntes
Digitala Projekt

Rickard Lövblom och Gunnar Ring

2 juni 2006

Abstract

This report covers the design of a microprocessor-based application. The prototype is a MIDI-controlled tone generetor, using the Yamaha YMF262 for sound generation with FM-synthesis. The system is based on the 32-bit asynchronous microprocessor Motorola 68008. An emulator for the 68008, it-68, was used for testing and troubleshooting during the development process. The prototype is capable of receiving MIDI-data and generating a monophonic tone according to the pitch and velocity specified in a MIDI-message. It also features a user interface consisting of eight buttons and a lcd-display. The essential parameters of the YMF262 are editable through this user interface.

Innehåll

1 Inledning

Kursen digitala projekt syftar till att utveckla en förståelse för industriellt utvecklingsarbete. En fungerande prototyp av en mikroprocessorstyrd applikation skall konstrueras under kursens gång. I detta specifika projekt konstrueras en prototyp av en MIDI-styrd tongenerator [3], vilken använder sig av Yamahas YMF262 (SOP IC) för ljudgenerering. YMF262 [1] återfinns bland annat på äldre SoundBlaster-kompatibla ISA-ljudkort och arbetar tillsammans med en för ändamålet specialkonstruerad DA-omvandlare, YAC512 [2]. I tabell 1 och tabell 2 presenteras översiktligt egenskaperna hos YMF262 respektive YAC512.

YMF262
FM-syntes
4 digitala audio-kanaler
8 olika vågformer
Upp till 4 operatorer (oscillatorer) per röst
Max 18 simultana röster
LFO
490 dataregister

Tabell 1: YMF262 översikt

YAC512
Seriell DA-omvandlare med s/h
16 bitars dynamiskt omfång

Tabell 2: YAC512 översikt

2 Kravspecifikation

Tongeneratorn skall konstrueras för att uppfylla kraven specificerade i 2.1 och 2.2.

2.1 Grundläggande krav

1. Enheten skall generera ljud med hjälp av YMF262 till en analog stereoutgång.

2. Enheten skall kunna styras externt via MIDI-kommunikation i realtid.

3. Instrument (ljudinställningar) skall kunna sparas permanent i ett lokalt minne. Minnet skall ha tillräcklig kapacitet för lagring av tio instrument.

2.2 Önskvärda egenskaper

1. Ljudgeneratorns parametrar (ljudets karaktär) skall kunna justeras externt via MIDI-kommunikation samt med hjälp av ett användargränssnitt på enheten.

2. En uppsättning av instrument i det lokala minnet skall kunna överföras med hjälp av MIDI SYSEX-meddelanden till en extern MIDI-enhet, exempelvis en persondator.

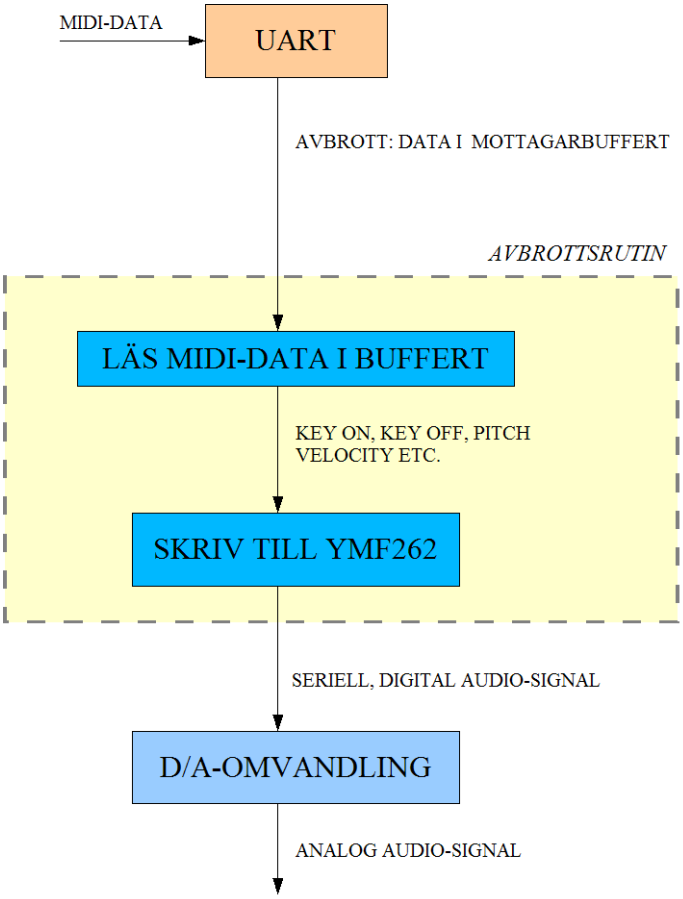
3. Tidsfördröjningen mellan mottagen MIDI-data och genererad ljudsignal på den analoga stereoutgången skall vara rimligt låg, max 20 ms.

4. Signal-brus-förhållandet för signalen på den analoga stereoutgången skall vara minst 30 dB.

5. Spänningsnivån hos signalen på den analoga stereoutgången skall följa standarden för linjenivå (+4 dBu).

3 Metod

Den grundläggande principen för systemet är att mottagen MIDI-data används för att styra YMF262, vilken därpå genererar en digital audio-signal. Denna digitala signal skall sedan D/A-omvandlas med hjälp av YAC512. Förloppet åskådliggörs i figur 1.



Figur 1: Blockschema för ljudgenerering

3.1 Hårdvarukonstruktion

Systemet (se C) är uppbyggt kring den asynkrona 32-bitars mikroprocessorn Motorola 68008, vilken är försedd med 8-bitars databuss, 1 MB adressrymd och två externa avbrottsingångar. Klocksignalen till 68008 genereras med hjälp av en extern oscillatorkrets på 8 MHz.

Arbets- och programminnet utgörs av ett SRAM, NEC 43256, respektive ett EPROM, 27C128. Vid testning och felsökning av hård- och mjukvara används en emulator för Motorola 68008, it-68.

3.1.1 Ljudgenerering

YMF262 genererar en fyrkanalig digital audio-signal. Signalen fördelas över två utgångar, med två kanaler på varje utgång. För att skilja de båda kanalerna på en utgång från varandra vid D/A-omvandling, används två samplingssignaler som ansluts till YAC512. De analoga utgångarna på YAC512

är kopplade till spänningsföljare.

YMF262 kräver en klocksignal med frekvensen 14,32 MHz, vilken genereras med hjälp av en extern oscillatorkrets.

3.1.2 MIDI-kommunikation

MIDI-kommunikation är en asynkron seriekommunikation med överföringshastigheten 31250 baud, 8-bitars ordlängd samt en start- och stoppbit. En UART, ST16C550, används för att ta emot MIDI-data. När data registreras i UART:ens mottagarbuffert genereras ett avbrott. För att undvika störningar placeras en optokopplare vid UART:ens seriella ingång.

3.1.3 Användargränssnitt

Användargränssnittet utgörs av en alfanumerisk display, SHARP LM162, samt åtta tryckknappar. LCD:n har två rader med vardera 16 tecken. De åtta tryckknapparna är anslutna till databussen via ett gränssnitt (74HC373 D-latch) som förser tryckknapparna med ett högimpedivt tillstånd då de inte används. När en knapp trycks ned genereras ett avbrott och ett värde kan läsas från databussen. De åtta tryckknapparna är via gränssnittet anslutna till var sin bit på databussen, vilket ger varje tryckknapp ett unikt värde vid avläsning.

3.1.4 Lagring av instrument

Instrumentdata lagras i ett EEPROM, AT28C64B.

3.1.5 Avbrottshantering

Avbrottshanteringen realiseras med hjälp av en programmerbar logikkrets, PALCE22V10 (se B.1), vilken genererar signaler till Motorola 68008:s avbrottsingångar. Systemet använder sig av två avbrott med olika prioritet, ett för mottagen indata i UART:ens mottagarbuffert (prioritet 1) och ett för nedtryckt knapp (prioritet 2).

3.1.6 Adressering

Adresseringen realiseras med hjälp av en programmerbar logikkrets, PALCE22V10 (se B.2), vilken styr chip select-signalerna till de olika periferienheterna. Adressrymden fördelas enligt tabell 3.

3.2 Programvara

Programkoden är skriven i ANSI C och har fördelats över filerna main.c (se A.1), ymf262.c (se A.2), menu.c (se A.3), midi.c (se A.4), btn.c (se A.5),

Periferienhet	Från	Till
EPROM:	00000	0FFFF
EEPROM:	10000	1FFFF
RAM:	30000	4FFFF
UART:	60000	60010
LCD:	61000	61010
YMF262:	62000	62010
TRYCKKNAPPAR:	63000	63010

Tabell 3: Addressrymd

display.c (se A.6) och instrument.c (se A.7). I header-filerna var.h (se A.8) och func.h (se A.9) deklareras globala variabler och funktioner.

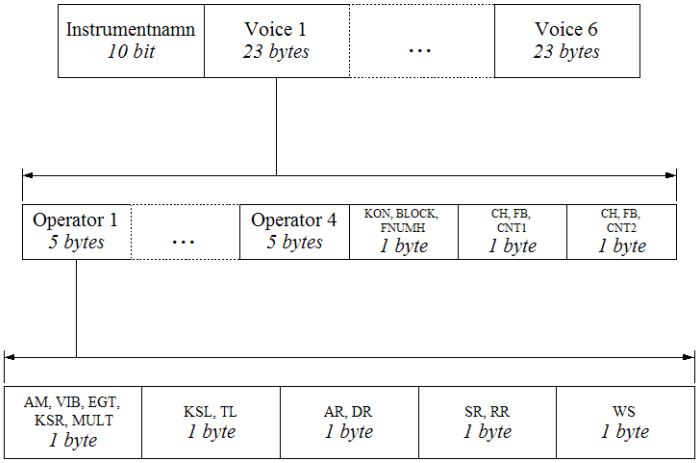
3.2.1 Huvudprogram - main.c

I huvudprogrammet anropas initieringsfunktioner, varpå processorn sätts i en evig loop i väntan på avbrott.

3.2.2 Styrning av YMF262 - ymf262.c

Vid ljudgenerering med YMF262 används operatorer (oscillatorer), vilka kan generera åtta olika vågformer. Fyra operatorer utgör en röst och sex röster utgör tillsammans ett komplett instrument. De olika rösterna i ett instrument har ingen inverkan på varandra. Däremot finns det ett antal parametrar som styr hur de fyra operatorerna i en röst skall arbeta tillsammans. Dessa parametrar är KON (key on/start sound generation), BLOCK (octave data), FNUMH (frequency number MSB), CH (channel), FB (feedback) och CNT (operator connection). Varje operator har i sin tur parametrarna AM (amplitude modulation), VIB (vibrato), KSR (key scale rate), EGT (envelope type), MULT (frequency data multiplier), KSL (key scale level), TL (total level), AR (attack rate), DR (decay rate), SR (sustain rate), RR (release rate) och WS (waveform select) som styr ljudets karaktär. För en utförlig beskrivning av de parametrar som används vid ljudgenerering, se datablad för YMF262 [1] .

Alla parametrar som tillhör ett instrument placeras i en vektor (se figur 2). Elementens ordning har anpassats för att utnyttja symmetrin i YMF262:s registerstruktur [1] . Algoritmerna har implementerats så att styrningen av YMF262 kan ske på tre olika nivåer: registernivå, operatornivå och instrumentnivå.



Figur 2: Struktur för instrumentdata

3.2.3 MIDI-kommunikation - midi.c

Vid initiering konfigueras UART:en för att följa MIDI-standardens dataöverföringsspecifikationer, dvs 31250 baud, 8-bitars ordlängd samt en start- och stoppbit. Vid avbrott från UART läses datan i dess mottagarbuffert. En SWITCH CASE-sats används för att identifiera MIDI-status bytes och MIDI-data bytes. Implementationen tillåter två olika typer av status bytes, NOTE ON och NOTE OFF, samt två typer av data bytes, PITCH NUMBER och VELOCITY.

3.2.4 Hantering av avbrott från tryckknappar - btn.c

En knapptryckning ger en insignal till menysystemets tillståndsmaskin. De tryckknappar som används är

1. MENY HÖGER
2. MENY VÄNSTER
3. MENY UPP
4. MENY NED
5. ENTER
6. EXIT
7. EDIT
8. SAVE

Riktningsknapparna HÖGER, VÄNSTER, UPP och NED används för navigering i menysystemet samt för justering av parametervärden för ljudgenerering. ENTER och EXIT används för att välja nästa respektive föregående meny (se A.5). SAVE sparar det aktuella instrumentet till EEPROM.

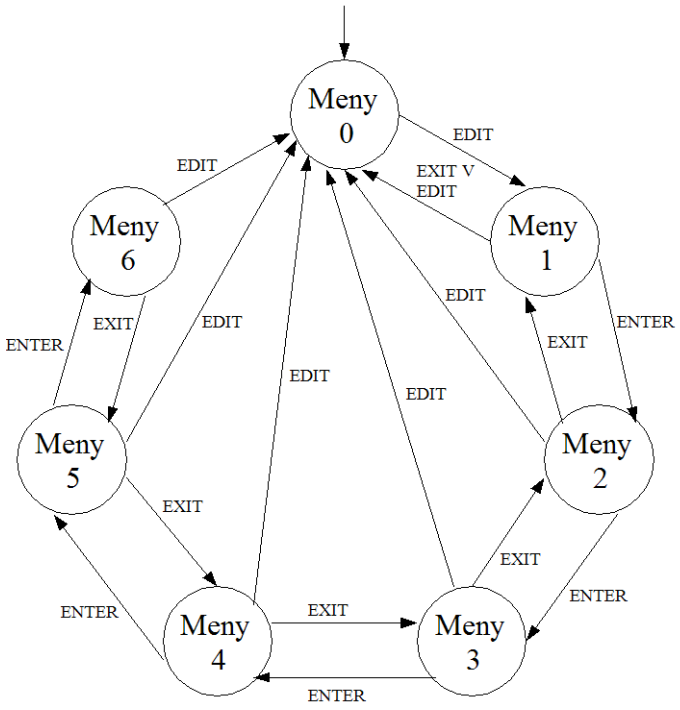
3.2.5 Menysystem - menu.c

Menysystemet utgörs av en tillståndsmaskin (se figur 3) bestående av sju tillstånd (menyer). De redigerbara parametrarna i menysystemet skiljer sig något från de som återfinns i vektorn för instrumentdata (se 3.2.2). Detta beror på att vissa funktioner är utspridda över ett flertal register hos YMF262 (exempelvis EGT och SR, där EGT måste aktiveras för att SR skall kunna användas). Varje element i vektorn för instrumentdata har storleken 1 byte, och kan bestå av ett antal sammansatta parametrar. För att möjliggöra redigering av enskilda parametrar måste de sammansatta parametrarna i ett element separeras. Detta sker genom användning av bitvis AND samt skiftning (se A.2). På motsvarande vis måste de separerade parametrarna sammansättas till element i vektorn för instrumentdata innan de kan skrivas till YMF262. Menysystemet har följande struktur.

Meny 0 Val av instrument

Meny 1 Redigiering - val av röst 1... 6

- **Meny 2** Gemensamma röstparametrar: OUT (kombination av CH för två operatorpar), OCT (BLOCK),
  FB, CNT (kombination av CNT för två operatorpar)
- **Meny 3** Val av operator 1... 4
  - **Meny 4** Envelope-parametrar: AT (AR), SUS (EGT och SR), DEC (DR), REL (RR)
  - **Meny 5** Modulation: AM, VIB, KSR, KSL
  - **Meny 6** Vågform, volym etc.: WAV (WS), MULT, TL



Figur 3: Tillståndsgraf för menysystem

Se A.3 och A.5 för implementation.

3.2.6 Styrning av display - display.c

Här finns funktioner som skriver teckensträngar med godtycklig längd till en valfri position på den alfanumeriska displayen.

3.2.7 Instrumenthantering - instrument.c

Vid hantering av instrument avnänds ett antal funktioner för att skriva skriva och läsa instrumentdata från EEPROM. Varje instrument kräver 148 byte lagringsutrymme, vilket motsvarar en vektor för instrumentdata (se 3.2.2). Instrumentdatan börjar på EEPROM:ets basadress och varje element i vektorn för instrumentdata lagras succesivt på följande adress.

4 Resultat och diskussion

Den färdiga prototypen kan ta emot MIDI-data och generera ljud utifrån denna. De viktigaste parametrarna hos YMF262 kan styras med hjälp av menysystemet. Dessutom kan instrument sparas och därefter hämtas från EEPROM. Detta innebär att de grundläggande kraven har uppfyllts, däremot har inte MIDI SYSEX-meddelanden implementerats. MIDI-implementationen

är inte fullständig, vilket begränsar möjligheterna att styra tongeneratorn från en extern MIDI-enhet. Störningsnivåer och tidsfördröjning har inte studerats.

Funktionaliteten begränsas av att instrumentnamn inte kan redigeras i menysystemet, vilket förmodligen är relativt enkelt att åtgärda. Ett annat problem är att parametern BLOCK används för att välja oktav vid röstredigering. Detta leder till att endast en oktav kan utnyttjas vid MIDI-styrning. En lösning till detta problem är att avlägsna parametern BLOCK från menysystemet och istället använda denna för att generera frekvensdata till YMF262 vid ljudgenerering.

Tongeneratorn är monofonisk, vilket beror på att samtliga sex röster genereras för samma grundfrekvens. En annorlunda mjukvaruimplementation skulle tillåta upp till sex rösters polyfoni, genom att låta endast en röst påverka ljudets karaktär. Då finns det möjlighet att simultant generera sex identiska röster på sex olika frekvenser.

Referenser

[1] “YMF262 FM Operator Type L3”, Catalog No. 4MF262A6, YAMAHA CORPORATION, Nov. 1994.

[2] “YAC512 2-Channel Floating D/A Converter”, Catalog No. LSI-4AC5124, YAMAHA CORPORATION, Apr. 1994.

[3] “MIDI Manufacturers Association”, www.midi.org, 2006.

A Programkod

A.1 main.c

```
#include "var.h"
#include "func.h"

main() {
        init_instruments();
        init_ymf();
        init_uart();
        display_init();
        init_menu();
        init_btn();
        load_instrument(0);
        load_menu();
        _avben();


        for(;;);
        return;

}

/* Button pressed interrupt */
void exp2(void) {
        btn_down();
}

/* UART buffer data available interrupt */
void exp5(void) {
        midi_in();
}
```

A.2 ymf262.c

```
/* Instrument structure: unsigned short int INSTRUMENT[] =
 [VOICE1 VOICE2 VOICE3 VOICE4 VOICE5 VOICE6]

 Voice structure: unsigned short int VOICE[] =
[OPERATOR1(0-4) OPERATOR2(5-9) OPERATOR3(10-14) OPERATOR4(15-19)
 KON_BLOCK_FNUMH(20) CH_FB_CNT1(21) CHN_FB_CNT2(22)]

 Operator structure: unsigned short int OPERATOR[] =
 [AM_VIB_EGT_KSR_MULT(0) KSL_TL(1) AR_DR(2) SL_RR(3) WS(4)]     */
```

```
#include "var.h"
#include "func.h"

#define YMF_WAIT 100

unsigned short int *ymf_addr0;          /* YMF-262 write address register array 0. */
unsigned short int *ymf_addr1;          /* YMF-262 write address register array 1. */
unsigned short int *ymf_data;           /* YMF-262 write data register 0. */

unsigned short int parameter_values[18];

static unsigned int voice_base_address[] = {0x20, 0x21, 0x22, 0x120, 0x121
, 0x122};
static unsigned int par_offset_addr[] = {0x0, 0x20, 0x40, 0x60, 0xC0, 0x3,
 0x23, 0x43, 0x63, 0xC3, 0x8, 0x28, 0x48, 0x68, 0xC8, 0xB, 0x2B, 0x4B, 0x6B,
 0xCB, 0x90, 0xA0, 0xA3};
static unsigned int fnuml_offset_addr = 0x80;
static unsigned short int key_on[] = {0, 0, 0, 0, 0, 0};
static unsigned int fnum[] = {0, 0, 0, 0, 0, 0};



/* Functions */

/* Initiate YMF262 */
void init_ymf(void);

/* Write to a register in th YMF262 */
static void ymfwreg(unsigned int, unsigned short int);

/* Delay between write cycles */
static void ymf_wait(void);

/* Write instrument */
void ymfwinstr(void);

/* Write operator */
void ymfwoperator(unsigned short int, unsigned short int);

/* Write shared voice parameters */
void ymfwshared(unsigned short int);
```

```
/* Get parameters from instrument data */
void set_par_values(unsigned short int, unsigned short int);

/* Update instrument data after operator parameter change */
void update_instr_data(void);

/* Generate sound */
void ymf_key_on(unsigned short int, unsigned int, unsigned short int);

/* Stop sound generation */
void ymf_key_off(unsigned short int);

/* Initiate YMF262 */
void init_ymf(void) {

        ymf_addr0 = (unsigned short int *) 0x0062000;
        ymf_addr1 = (unsigned short int *) 0x0062002;
        ymf_data = (unsigned short int *) 0x0062003;

        ymfwreg(0x1, 0x20);             /* Reset test register, enable ws */
        ymfwreg(0x101, 0x20);   /* Reset test register, enable ws */
        ymfwreg(0x105, 0x1);    /* Enable OPL3 */
        ymfwreg(0x104, 0x3F);   /* Enable 4 operators on all channels */
        ymfwreg(0x8, 0x0);              /* NTS = 0 => FNUM_MSB don't care */
        ymfwreg(0xBD, 0x0);     /* Rythm mode off */

}

/* Write to a register in th YMF262 */
static void ymfwreg(unsigned int address, unsigned short int data) {

        if (address >= 0x100) {

                ymf_wait();
                *ymf_addr1 = (unsigned short int) (address - 0x100);
                ymf_wait();
                *ymf_data = data;

        } else {

                ymf_wait();
                *ymf_addr0 = address;
                ymf_wait();
                *ymf_data = data;
```

```
        }

}

/* Delay between write cycles */
static void ymf_wait(void) {

        int i = 0;

        while (i < YMF_WAIT) {

                i++;

        }

}

/* Write instrument */
void ymfwinstr(void){

        unsigned short int voice, op;
        for (voice = 1; voice <= NBROF_VOICES; voice++){
                for (op = 1; op <= NBROF_OPS; op++){
                        ymfwoperator(voice, op);
                }
                ymfwshared(voice);
        }
}

/* Write operator */
void ymfwoperator(unsigned short int voice, unsigned short int op){

        unsigned int op_offset = ((unsigned int) op - 1) * NBROF_OP_PARS,
voice_offset = ((unsigned int) voice - 1) * VOICE_LENGTH, i;

        for (i = 0; i <= NBROF_OP_PARS; i++) {

                ymfwreg(voice_base_address[voice - 1] +
par_offset_addr[op_offset + i], instr_data[voice_offset + op_offset + i]);

        }

}
```

```
/* Write shared voice parameters */
void ymfwshared(unsigned short int voice){

        unsigned int voice_offset = ((unsigned int) voice - 1) *
VOICE_LENGTH, i;

        for (i = SHARED_PARS_OFFSET; i < SHARED_PARS_OFFSET +
NBROF_SHARED_PARS; i++) {

                ymfwreg(voice_base_address[voice - 1] +
par_offset_addr[i], instr_data[voice_offset + i]);

        }

}

/* Get parameters from instrument data */
void set_par_values(unsigned short int voice, unsigned short int op) {

        unsigned int voice_offset = ((unsigned int) voice - 1) *
VOICE_LENGTH, op_offset = ((unsigned int) op - 1) * NBROF_OP_PARS;

        /* Shared voice parameters */
        unsigned short int out, oct, fb, cnt;

        /* Operator specific parameters */
        unsigned short int at, sus, dec, rel, am, vib, ksr, ksl, wave,
tl, mult;

        /* Register block variables */
        unsigned short int am_vib_egt_ksr_mult, ksl_tl, ar_dr, sl_rr,
 ws, ch_fb_cnt1, ch_fb_cnt2, kon_block_fnumh, egt;

        /* Retrieve register block data */
        am_vib_egt_ksr_mult = instr_data[voice_offset + op_offset];
        ksl_tl = instr_data[voice_offset + op_offset + 1];
        ar_dr = instr_data[voice_offset + op_offset + 2];
        sl_rr = instr_data[voice_offset + op_offset + 3];
        ws = instr_data[voice_offset + op_offset + 4];
        kon_block_fnumh = instr_data[voice_offset + 20];
        ch_fb_cnt1 = instr_data[voice_offset + 21];
        ch_fb_cnt2 = instr_data[voice_offset + 22];

        /* Separate operator specific parameters */
```

```
        am = (am_vib_egt_ksr_mult & 0x80)/0x80;
        vib = (am_vib_egt_ksr_mult & 0x40)/0x40;
        egt = (am_vib_egt_ksr_mult & 0x20)/0x20;
        ksr = (am_vib_egt_ksr_mult & 0x10)/0x10;
        mult = am_vib_egt_ksr_mult & 0xF;
        ksl = (ksl_tl & 0xC0)/0x40;
        tl = ksl_tl & 0x3F;
        at = (ar_dr & 0xF0)/0x10;
        dec = ar_dr & 0xF;
        sus = ((sl_rr & 0xF0)/0x10) & 0xF * egt;
        rel = sl_rr & 0xF;
        wave = ws & 0x7;

        /* Separate shared instrument parameters */
        oct = (kon_block_fnumh & 0x1C)/0x4;
        out = (ch_fb_cnt1 & 0x30)/0x10;
        fb = (ch_fb_cnt1 & 0xE)/0x2;
        cnt = (ch_fb_cnt1 & 0x1) * 0x2 + (ch_fb_cnt2 & 0x1);

        /* Update parameter values */
        parameter_values[1] = voice;
        parameter_values[2] = out;
        parameter_values[3] = oct;
        parameter_values[4] = fb;
        parameter_values[5] = cnt;
        parameter_values[6] = op;
        parameter_values[7] = at;
        parameter_values[8] = sus;
        parameter_values[9] = dec;
        parameter_values[10] = rel;
        parameter_values[11] = am;
        parameter_values[12] = vib;
        parameter_values[13] = ksr;
        parameter_values[14] = ksl;
        parameter_values[15] = wave;
        parameter_values[16] = mult;
        parameter_values[17] = tl;

}

/* Update instrument data after operator parameter change */
void update_instr_data(void) {

        unsigned int voice_offset, op_offset;
```

```
	/* Shared voice parameters */
	unsigned short int out, oct, fb, cnt, kon, fnumber;

	/* Operator specific parameters */
	unsigned short int at, sus, dec, rel, am, vib, ksr, ksl, wave,
tl, mult;

	/* Register block variables */
	unsigned short int am_vib_egt_ksr_mult, ksl_tl, ar_dr, sl_rr,
ws, ch_fb_cnt1, ch_fb_cnt2, kon_block_fnumh, egt, voice, op;

	voice = parameter_values[1];
	out = parameter_values[2];
	oct = parameter_values[3];
	fb = parameter_values[4];
	cnt = parameter_values[5];
	op = parameter_values[6];
	at = parameter_values[7];
	sus = parameter_values[8];
	dec = parameter_values[9];
	rel = parameter_values[10];
	am = parameter_values[11];
	vib = parameter_values[12];
	ksr = parameter_values[13];
	ksl = parameter_values[14];
	wave = parameter_values[15];
	mult = parameter_values[16];
	tl = parameter_values[17];

	kon = key_on[voice];
	fnumber = fnum[voice];

	if(sus > 0)
		egt = 1;
	else
		egt = 0;

	am_vib_egt_ksr_mult = am * 0x80 + vib * 0x40 + egt * 0x20 +ksr *
0x10 + mult;
	ksl_tl = ksl * 0x40 + tl;
	ar_dr = at * 0x10 + dec;
	sl_rr = sus * 0x10 + rel;
	ws = wave;
```

```
        kon_block_fnumh = kon * 0x20 + oct * 0x4 + (fnumber & 0x100)/0x100;
/* NTS = 0 => FNUM_MSB don't care */
        ch_fb_cnt1 = out * 0x10 + fb * 0x2 + (cnt & 0x2)/0x2;
        ch_fb_cnt2 = out * 0x10 + fb * 0x2 + (cnt & 0x1);

        voice_offset = ((unsigned int) voice - 1) * VOICE_LENGTH,
 op_offset = ((unsigned int) op - 1) * NBROF_OP_PARS;

        instr_data[voice_offset + op_offset] = am_vib_egt_ksr_mult;
        instr_data[voice_offset + op_offset + 1] = ksl_tl;
        instr_data[voice_offset + op_offset + 2] = ar_dr;
        instr_data[voice_offset + op_offset + 3] = sl_rr;
        instr_data[voice_offset + op_offset + 4] = ws;
        instr_data[voice_offset + SHARED_PARS_OFFSET] = kon_block_fnumh;
        instr_data[voice_offset + SHARED_PARS_OFFSET + 1] = ch_fb_cnt1;
        instr_data[voice_offset + SHARED_PARS_OFFSET + 2] = ch_fb_cnt2;

}

/* Generate sound */
void ymf_key_on(unsigned short int voice, unsigned int fnumber,
 unsigned short int velocity) {

        unsigned short int kon_block_fnumh, new_kon_block_fnumh,
 ksl_tl, tl, new_ksl_tl, op;
        unsigned int voice_offset = ((unsigned int) voice - 1) *
 VOICE_LENGTH, op_offset;

        key_on[voice] = 1;
        fnum[voice] = fnumber;

        for (op = 1; op <= NBROF_OPS; op++) {
                op_offset = ((unsigned int) op - 1) * NBROF_OP_PARS;
                ksl_tl = instr_data[voice_offset + op_offset + 1];
                tl = ksl_tl & 0x3F;
                new_ksl_tl = tl * velocity;
                ymfwreg(voice_base_address[voice - 1] +
 par_offset_addr[op_offset + 1], new_ksl_tl);           /* Scale tl with velocity */
        }


        kon_block_fnumh = instr_data[voice_offset + SHARED_PARS_OFFSET];
        /*ksl_tl = instr_data[voice_offset + op_offset + 1];*/
        ymfwreg(voice_base_address[voice - 1] + fnuml_offset_addr,
```

```
 fnumber & 0xFF);                                                    /* Set fnuml */
        new_kon_block_fnumh = 0x20 + (kon_block_fnumh & 0x1C) +
(fnumber & 0x100)/0x100;
/* Set fnumh and kon */
        ymfwreg(voice_base_address[voice - 1] +
par_offset_addr[SHARED_PARS_OFFSET], new_kon_block_fnumh);


}

/* Stop sound generation */
void ymf_key_off(unsigned short int voice) {

        unsigned short int kon_block_fnumh, block;
        unsigned int voice_offset = ((unsigned int) voice - 1) *
VOICE_LENGTH;

        key_on[voice] = 0;

        kon_block_fnumh = instr_data[voice_offset + SHARED_PARS_OFFSET];
        block = kon_block_fnumh & 0x1C;
        ymfwreg(voice_base_address[voice - 1] +
par_offset_addr[SHARED_PARS_OFFSET], block);
        ymfwreg(voice_base_address[voice - 1] +
fnuml_offset_addr, 0x0);
                                                        /* Set fnuml to 0 */


}
```

A.3 menu.c

```
#include <stdio.h>
#include "var.h"
#include "func.h"


unsigned short int par_display_index[][] =  {
{0, 0, 0, 0, 0, 0},
{2, 4, 6, 8, 10, 12},
{0, 4, 8, 11, 0, 0},
{3, 6, 9, 12, 0, 0},
{0, 3, 7, 11, 0, 0},
```

```
{0, 3, 7, 11, 0, 0},
{0, 4, 9, 0, 0, 0}
};

unsigned short int max_par_index[] = {NBROF_INSTR - 1, 5, 3, 3, 3, 3, 2};
unsigned short int menu_par_offset[] = {0, 1, 2, 6, 7, 11, 15};
unsigned short int max_par_value[] = {0, 0, 3, 7, 7, 3, 0, 15, 15, 15, 15, 1, 1, 1, 3
unsigned short int menu = 0, sel_par = 0;




/* Initiate menu */
void init_menu(void);

/* Load menu */
void load_menu(void);

/* Update menu */
void update_menu(void);

/* Initiate menu */
void init_menu(void) {
}

/* Load menu */
void load_menu(void) {

char menu_titles[][] = {
{'I', 'N', 'S', 'T', 'R', 'U', 'M', 'E', 'N', 'T', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
{' ', ' ', ' ', ' ', ' ', ' ', 'V', 'O', 'I', 'C', 'E', ' ', ' ', ' ', ' ', ' ', ' '},
{'O', 'U', 'T', ' ', ' ', 'O', 'C', 'T', ' ', 'F', 'B', ' ', 'C', 'N', 'T', ' ', ' '},
{' ', ' ', ' ', ' ', ' ', 'O', 'P', 'E', 'R', 'A', 'T', 'O', 'R', ' ', ' ', ' ', ' '},
{'A', 'T', ' ', 'S', 'U', 'S', ' ', 'D', 'E', 'C', ' ', 'R', 'E', 'L', ' ', ' ', ' '},
{'A', 'M', ' ', 'V', 'I', 'B', ' ', 'K', 'S', 'R', ' ', 'K', 'S', 'L', ' ', ' ', ' '},
{'W', 'A', 'V', ' ', 'M', 'U', 'L', 'T', ' ', 'T', 'L', ' ', ' ', ' ', ' ', ' ', ' '}
};

char title[16], instr_nr_char[2];
char string1[] = {' ', ' ', '1', ' ', '2', ' ', '3', ' ', '4', ' ', '5', ' ', '6', '
char string3[] = {' ', ' ', ' ', '1', ' ', ' ', '2', ' ', ' ', '3', ' ', ' ', '4', '
char blankspace[] = {' '};

unsigned short int i, instr_nr, text_len;
```

```
display_clear();
hide_cursor();

for (i = 0; i < 16; i++) {
title[i] = menu_titles[menu][i];
}
display_upper(title, 16, 0);

if ((menu != 0) && (menu != 1) && (menu != 3)) {

for (sel_par = 0; sel_par <= max_par_index[menu]; sel_par++) {
update_menu();
}
set_cursor(par_display_index[menu][0]);

} else if (menu == 0) {

instr_nr = parameter_values[0];
/*itoa((int) instr_nr, instr_nr_char, 10);*/
sprintf(instr_nr_char, "%d", (int) instr_nr);
if (instr_nr > 9)
text_len = 2;
else
text_len = 1;
display_lower(instr_nr_char, text_len, 0);
display_lower(blankspace, 1, text_len);
display_lower(instr_name, 10, text_len + 1);

} else if (menu == 1) {
display_lower(string1, 16, 0);
set_cursor(par_display_index[menu][0]);

} else if (menu == 3) {
display_lower(string3, 16, 0);
set_cursor(par_display_index[menu][0]);
}
sel_par = 0;
}

/* Update menu */
void update_menu(void) {
```

```
char par_value_char[2];
unsigned short int text_len, par_value, index;
par_value = parameter_values[menu_par_offset[menu] + sel_par];
sprintf(par_value_char, "%d", (int) par_value);
/*itoa((int) par_value, par_value_char, 10);*/
if (par_value > 9)
text_len = 2;
else
text_len = 1;
index = par_display_index[menu][sel_par];
display_lower(par_value_char, text_len, index);
}
```

A.4 midi.c

```
#include "var.h"
#include "func.h"*/

#define IDLE 0
#define NOTE_ON1 1
#define NOTE_OFF1 3
#define NOTE_ON2 2
#define NOTE_OFF2 4

unsigned short int *uart_rbr_thr_dll;
/* Reciever buffer (read, DLAB=0), transmitter holding (write, DLAB=0),
 divisor latch LS (DLAB=1) registers. */
unsigned short int *uart_ier_dlm;
/* Interrupt enable (DLAB=0), Divisor latch MS (DLAB=1) registers. */
unsigned short int *uart_iir_fcr;
/* Interrupt identify (read), FIFO control (write) registers. */
unsigned short int *uart_lcr;                   /* Line control register. */
unsigned short int *uart_mcr;                   /* Modem control register. */
unsigned short int *uart_lsr;                   /* Line status register. */
unsigned short int *uart_msr;                   /* Modem status register. */
unsigned short int *uart_scr;                   /* Scratch register. */

unsigned short int midi_data;
unsigned short int midi_status;
unsigned short int pitch_number;
unsigned short int velocity;

/* Initiate UART */
void init_uart(void);
```

```
void midi_in(void);

/* Initiate UART */
void init_uart(void) {

        uart_rbr_thr_dll = (unsigned short int *) 0x0060000;
        uart_ier_dlm = (unsigned short int *) 0x0060001;
        uart_iir_fcr = (unsigned short int *) 0x0060002;
        uart_lcr = (unsigned short int *) 0x0060003;
        uart_mcr = (unsigned short int *) 0x0060004;
        uart_lsr = (unsigned short int *) 0x0060005;
        uart_msr = (unsigned short int *) 0x0060006;
        uart_scr = (unsigned short int *) 0x0060007;


        *uart_lcr = 0x0000080;                  /* Set DLAB=1 */
        *uart_rbr_thr_dll = 0x0000010;  /* Divisor latch LS => 31250 baud */
        *uart_ier_dlm = 0x00000000;     /* Divisor lacth MS = 0 */
        *uart_lcr = 0x0000003;
/* Set DLAB=0, 8-bit word, 1 stop bit, no parity, no break */
        *uart_mcr = 0x0000000;                  /* Reset mcr */
        *uart_iir_fcr = 0x0000007;
/* Enable FIFO, clear FIFO, reciever buffer trigger level = 1 byte */
        *uart_ier_dlm = 0x0000001;
/* Enable recieved data available interrupt. */

        midi_status = IDLE;

}

void midi_in(void) {
        unsigned short int oktav;
        unsigned short int ton;
        unsigned int fnumber;
        float frek_data[]= {261.625565,277.1818210949,293.6642316899,
311.12773815365,329.6272818548,349.234286572595,369.99349027865,
391.99620029515,415.304421881,439.99925896135,466.164431717,493.88366032875};



        midi_data = *uart_rbr_thr_dll;
```

```
        switch(midi_status) {
                                case NOTE_ON1:
                                        pitch_number = midi_data;
                                        midi_status = NOTE_ON2;
                                        break;
                                case NOTE_ON2:
                                        velocity = midi_data/127;
                                                                                oktav
                                        ton = pitch_number%12;


                                /*      frek = 261*2^oktav*2^(ton/oktav);
                                        fnumber=((unsigned int)frek)10;
                                        fnumber=pitch_number; */

                                        fnumber=(unsigned int)(frek_data[ton]*
0.329375);



                                        ymf_key_on(1, fnumber,velocity);
                        /*              ymf_key_on(2, fnumber, velocity);
                                        ymf_key_on(3, fnumber, velocity);
                                        ymf_key_on(4, fnumber, velocity);
                                        ymf_key_on(5, fnumber, velocity);
                                        ymf_key_on(6, fnumber, velocity); */
                                        midi_status = IDLE;
                                        break;
                                case NOTE_OFF1:
                                        midi_status = NOTE_OFF2;
                                        break;
                                case NOTE_OFF2:
                                        ymf_key_off(1);
                                        ymf_key_off(2);
                                        ymf_key_off(3);
                                        ymf_key_off(4);
                                        ymf_key_off(5);
                                        ymf_key_off(6);
                                        midi_status = IDLE;
                                        break;
                                case IDLE:
                                        if(midi_data >= 0x80 ) {
                                                switch (midi_data & 0xF0) {
```

```
                                                        case 0x90:

midi_status = NOTE_ON1;
                                                                break;
                                                        case 0x80:

midi_status = NOTE_OFF1;
                                                                break;
                                                        default:
                                                                break;
                                                }
                                        }
                                        break;
                                default:
                                        break;
        }

}
```

A.5 btn.c

```
#include "var.h"
#include "func.h"

/* Button constants */
#define R_BTN 0x1
#define L_BTN 0x2
#define U_BTN 0x4
#define D_BTN 0x8
#define EN_BTN 0x10
#define EX_BTN 0x20
#define ED_BTN 0x40
#define S_BTN 0x80

unsigned short int *btn_data;

/* Initiate buttons */
void init_btn(void);

/* Button pressed */
void btn_down(void);

/* Right button pressed */
static void r_btn_down(void);
```

```
/* Left button pressed */
static void l_btn_down(void);

/* Up button pressed */
static void u_btn_down(void);

/* Down button pressed */
static void d_btn_down(void);

/* Enter button pressed */
static void en_btn_down(void);

/* Exit button pressed */
static void ex_btn_down(void);

/* Edit button pressed */
static void ed_btn_down(void);

/* Save button pressed */
static void s_btn_down(void);


/* Initiate buttons */
void init_btn(void) {

        btn_data = (unsigned short int *) 0x0063000;

}

/* Button pressed */
void btn_down(void) {

        unsigned short int btn = *btn_data;

        switch (btn) {
                case R_BTN: r_btn_down(); break;
                case L_BTN: l_btn_down(); break;
                case U_BTN: u_btn_down(); break;
                case D_BTN: d_btn_down(); break;
                case EN_BTN: en_btn_down(); break;
                case EX_BTN: ex_btn_down(); break;
                case ED_BTN: ed_btn_down(); break;
                case S_BTN: s_btn_down(); break;
```

```
        }

}

/* Right button pressed */
static void r_btn_down(void) {

        if((menu > 0) && (sel_par < max_par_index[menu])) {

                sel_par++;
                set_cursor(par_display_index[menu][sel_par]);

                 /*Instrument select menu*/
        } else if((menu == 0) && (parameter_values[0] < NBROF_INSTR)) {

                parameter_values[0]++;
                load_instrument(parameter_values[0]);
                load_menu();

        }


}

/* Left button pressed */
static void l_btn_down(void) {

        if((menu > 0) && (sel_par > 0)) {

                sel_par--;
                set_cursor(par_display_index[menu][sel_par]);

        } else if((menu == 0) && (parameter_values[0] > 0)) {

                parameter_values[0]--;
                load_instrument(parameter_values[0]);
                load_menu();
        }

}

/* Up button pressed */
static void u_btn_down(void) {
```

```
        unsigned short int curr_par_index = menu_par_offset[menu] + sel_par;

        if ((menu != 0) && (menu != 1) && (menu != 3) &&
 (parameter_values[curr_par_index] < max_par_value[curr_par_index])) {
                        hide_cursor();
                        parameter_values[curr_par_index]++;
                        update_instr_data();
                        ymfwoperator(parameter_values[1], parameter_values[6]);
                        ymfwshared(parameter_values[1]);
                        update_menu();
                        set_cursor(par_display_index[menu][sel_par]);

        }

}

/* Down button pressed */
static void d_btn_down(void) {

        unsigned short int curr_par_index = menu_par_offset[menu] + sel_par;

        if ((menu != 0) && (menu != 1) && (menu != 3)
&& (parameter_values[curr_par_index] > 0)) {
                        hide_cursor();
                        parameter_values[curr_par_index]--;
                        update_instr_data();
                        ymfwoperator(parameter_values[1], parameter_values[6]);
                        ymfwshared(parameter_values[1]);
                        update_menu();
                        set_cursor(par_display_index[menu][sel_par]);

        }

}

/* Enter button pressed */
static void en_btn_down(void) {

        if ((menu != 0) && (menu != 1) && (menu != 3) && (menu < LAST_MENU)) {

                menu++;
                load_menu();
```

```
        } else if (menu == 1) {

                parameter_values[1] = sel_par + 1;
                parameter_values[6] = 1;
                set_par_values(parameter_values[1], parameter_values[6]);
                menu++;
                load_menu();

        } else if (menu == 3) {

                parameter_values[6] = sel_par + 1;
                set_par_values(parameter_values[1], parameter_values[6]);
                menu++;
                load_menu();

        }

}

/* Exit button pressed */
static void ex_btn_down(void) {

        if (menu > 0) {
                menu--;
                set_par_values(parameter_values[1], parameter_values[6]);
                load_menu();
        }

}

/* Edit button pressed */
static void ed_btn_down(void) {

        if (menu == 0) {
                menu++;
                /*set_par_values(parameter_values[1], parameter_values[6]);*/
                load_menu();

        } else {

                menu = 0;
                load_menu();

        }
```

```
}

/* Save button pressed */
static void s_btn_down(void) {

        save_instrument(parameter_values[0]);

}
```

A.6 display.c

```
#include "var.h"
#include "func.h"

#define DISPLAY_WAIT 1000

unsigned short int *display_instr;
unsigned short int *display_data;

/* Initiate display */
void display_init(void);

/* Clear entire display */
void display_clear(void);

/* Position and display cursor at given index on bottom line */
void set_cursor(unsigned short int);

/* Hide cursor */
void hide_cursor(void);

/* Display text string on first line and add blank space */
void display_upper(char[], unsigned short int, unsigned short int);

/* Display text string on second line and add blank space */
void display_lower(char[], unsigned short int, unsigned short int);

/* Write instruction */
static void display_wrinstr(unsigned short int);

/* Write data */
static void display_wrdata(char);
```

```
/* Delay between writes */
static void display_wait(void);

/* Initiate display */
void display_init(void) {

        display_instr = (unsigned short int *) 0x0061000;
        display_data = (unsigned short int *) 0x0061001;

        display_wrinstr(0xF);
        display_wrinstr(0x1);   /* Display clear */
        display_wrinstr(0x38);  /* 8-bit data interface, dual line */
        display_wrinstr(0x6);   /* Cursor increment mode, display shift off */
        display_wrinstr(0xC);   /* Display on, cursor off, blink off */

}

/* Clear entire display */
void display_clear(void) {

        display_wrinstr(0x1);

}

/* Position and display cursor at given index on bottom line */
void set_cursor(unsigned short int index) {

        unsigned short int i;

        hide_cursor();          /* Cursor off */
        display_wrinstr(0xC0);
/* Set DD RAM Address to first position on second line */

        for (i = 0; i < index; i++) {
                display_wrinstr(0x14); /* Move cursor to the right */
        }

        display_wrinstr(0xE); /* Cursor on */

}

/* Hide cursor */
void hide_cursor(void) {
```

```
        display_wrinstr(0xC);

}

/* Display text string on first line and add blank space */
void display_upper(char text[], unsigned short int text_len,
 unsigned short int index){

        unsigned short int i;

        display_wrinstr(0x80);
 /* Set DD RAM Address to first position on first line */

        for (i = 0; i < index; i++) {
                display_wrinstr(0x14); /* Move cursor to the right */
        }

        for (i = 0; i < text_len; i++) {
                display_wrdata(text[i]);
        }

        display_wrdata(' ');

}

/* Display text string on second line and add blank space */
void display_lower(char text[], unsigned short int text_len,
 unsigned short int index) {

        unsigned short int i;

        display_wrinstr(0xC0);
/* Set DD RAM Address to first position on second line */

        for (i = 0; i < index; i++) {
                display_wrinstr(0x14); /* Move cursor to the right */
        }

        for (i = 0; i < text_len; i++) {
                display_wrdata(text[i]);
        }

        display_wrdata(' ');
```

```
}

/* Write instruction */
static void display_wrinstr(unsigned short int instruction) {

        display_wait();
        *display_instr = instruction;

}

/* Write data */
static void display_wrdata(char data) {

        display_wait();
        *display_data = data;

}

/* Delay between writes */
static void display_wait(void) {

        int i = 0;
        while (i < DISPLAY_WAIT) {
                i++;
        }

}
```

A.7 instrument.c

```
#define VOICE_LENGTH 23
#define NBROF_OPS 4
#define NBROF_VOICES 6
#define NBROF_OP_PARS 5
#define NBROF_SHARED_PARS 3
#define SHARED_PARS_OFFSET 20
#define NBROF_INSTR 10
#define NAME_LENGTH 10
#define INSTR_SIZE 148

unsigned short int parameter_values[18];
unsigned short int instr_data[INSTR_SIZE - NAME_LENGTH];
char instr_name[NAME_LENGTH];
unsigned short int menu_par_offset[7] = {0, 1, 2, 6, 7, 11, 15};
```

```
unsigned short int max_par_value[18] = {0, 0, 3, 7, 7, 3, 0, 15, 15,
15, 15, 1, 1, 1, 3, 7, 15, 63};
unsigned short int menu = 0, sel_par;
unsigned short int max_par_index[7] = {NBROF_INSTR - 1, 5, 3, 3, 3, 3, 2};


void initt(void) {

/*menu_par_offset = {0, 1, 2, 6, 7, 11, 15};
max_par_value = {0, 0, 3, 7, 7, 3, 0, 15, 15, 15, 15, 1, 1, 1, 3, 7, 15, 63};*/
menu = 0;
sel_par = 0;
/*max_par_index = {NBROF_INSTR - 1, 5, 3, 3, 3, 3, 2}; */

}#include "var.h"
#include "func.h"

#define EEPROM_WRITE_WAIT 8000
#define EEPROM_READ_WAIT 1200
#define EEPROM_BASE_ADDRESS 0x10000


unsigned short int instr_data[INSTR_SIZE - NAME_LENGTH];

char instr_name[NAME_LENGTH];

/* Initiate instruments */
void init_instruments(void);

/* Load instrument name and data from EEPROM */
void load_instrument(unsigned short int);

/* Save instrument name and data to EEPROM */
void save_instrument(unsigned short int);

/* Load instrument data from EEPROM */
static void load_instr_data(unsigned long int);

/* Save instrument data to EEPROM */
static void save_instr_data(unsigned long int);

/* Load instrument name from EEPROM */
static void load_instr_name(unsigned long int);
```

```
/* Save instrument name to EEPROM */
static void save_instr_name(unsigned long int);

/* Write data to EEPROM */
static void wr_eeprom(unsigned long int, unsigned short int);

/* Read data from EEPROM */
static unsigned short int rd_eeprom(unsigned long int);

/* Delay between writes */
static void wr_wait(void);

/* Delay between reads */
static void rd_wait(void);

/* Initiate instruments */
void init_instruments(void) {

}

/* Load instrument name and data from EEPROM */
void load_instrument(unsigned short int instrument) {
        unsigned long int instr_base_address = (unsigned long int)
instrument * INSTR_SIZE;
        load_instr_data(instr_base_address);
        load_instr_name(instr_base_address);
        parameter_values[0] = instrument;
        ymfwinstr();
}

/* Save instrument name and data to EEPROM */
void save_instrument(unsigned short int instrument) {
        unsigned long int instr_base_address = (unsigned long int)
instrument * INSTR_SIZE;
        save_instr_data(instr_base_address);
        save_instr_name(instr_base_address);
}

/* Load instrument data from EEPROM */
static void load_instr_data(unsigned long int instr_base_address) {

        unsigned long int curr_address;
        int i;
```

```
        for(i = NAME_LENGTH; i < INSTR_SIZE; i++){
                curr_address = instr_base_address + i;
                instr_data[i - NAME_LENGTH] = rd_eeprom(curr_address);
        }

}

/* Save instrument data to EEPROM */
static void save_instr_data(unsigned long int instr_base_address) {

        unsigned long int curr_address;
        int i;

        for(i = NAME_LENGTH; i < INSTR_SIZE; i++){
                curr_address = instr_base_address + i;
                wr_eeprom(curr_address, instr_data[i - NAME_LENGTH]);
        }
```

A.8 var.h

```
#define VOICE_LENGTH 23
#define NBROF_OPS 4
#define NBROF_VOICES 6
#define NBROF_OP_PARS 5
#define NBROF_SHARED_PARS 3
#define SHARED_PARS_OFFSET 20
#define NBROF_INSTR 10
#define NAME_LENGTH 10
#define INSTR_SIZE 148
#define LAST_MENU 6

extern unsigned short int parameter_values[];
extern unsigned short int instr_data[];
extern char instr_name[];
extern unsigned short int par_display_index[7][6];
extern unsigned short int max_par_index[];
extern unsigned short int menu_par_offset[];
extern unsigned short int max_par_value[];
extern unsigned short int menu;
extern unsigned short int sel_par;
```

A.9 func.h

```
/* Initiate YMF262 */
void init_ymf(void);

/* Write instrument */
void ymfwinstr(void);

/* Write operator */
void ymfwoperator(unsigned short int, unsigned short int);

/* Write shared voice parameters */
void ymfwshared(unsigned short int);

/* Get parameters from instrument data */
void set_par_values(unsigned short int, unsigned short int);

/* Update instrument data after operator parameter change */
void update_instr_data(void);

/* Generate sound */
void ymf_key_on(unsigned short int, unsigned int, unsigned short int);

/* Stop sound generation */
void ymf_key_off(unsigned short int);

/* Initiate display */
void display_init(void);

/* Clear entire display */
void display_clear(void);

/* Position and display cursor at given index on bottom line */
void set_cursor(unsigned short int);

/* Hide cursor */
void hide_cursor(void);

/* Display text string on first line and add blank space */
void display_upper(char[], unsigned short int, unsigned short int);

/* Display text string on second line and add blank space */
void display_lower(char[], unsigned short int, unsigned short int);
```

```
/* Initiate instruments */
void init_instruments(void);

/* Load instrument name and data from EEPROM */
void load_instrument(unsigned short int);

/* Save instrument name and data to EEPROM */
void save_instrument(unsigned short int);

/* Initiate UART */
void init_uart(void);

/* Initiate buttons */
void init_btn(void);

/* Button pressed */
void btn_down(void);

/* Initiate menu */
void init_menu(void);

/* Load menu */
void load_menu(void);

/* Update menu */
void update_menu(void);

void midi_in(void);
```

B PAL

B.1 Avbrottshantering

```
Title OPL3Synth
Pattern Interrupt
Revision 0.0
Author R. Lovblom, G. Ring
Company LTH
Date 02-02-06
device 22v10
CLK 1
FC1 2 'Processorstatus
FC0 3
```

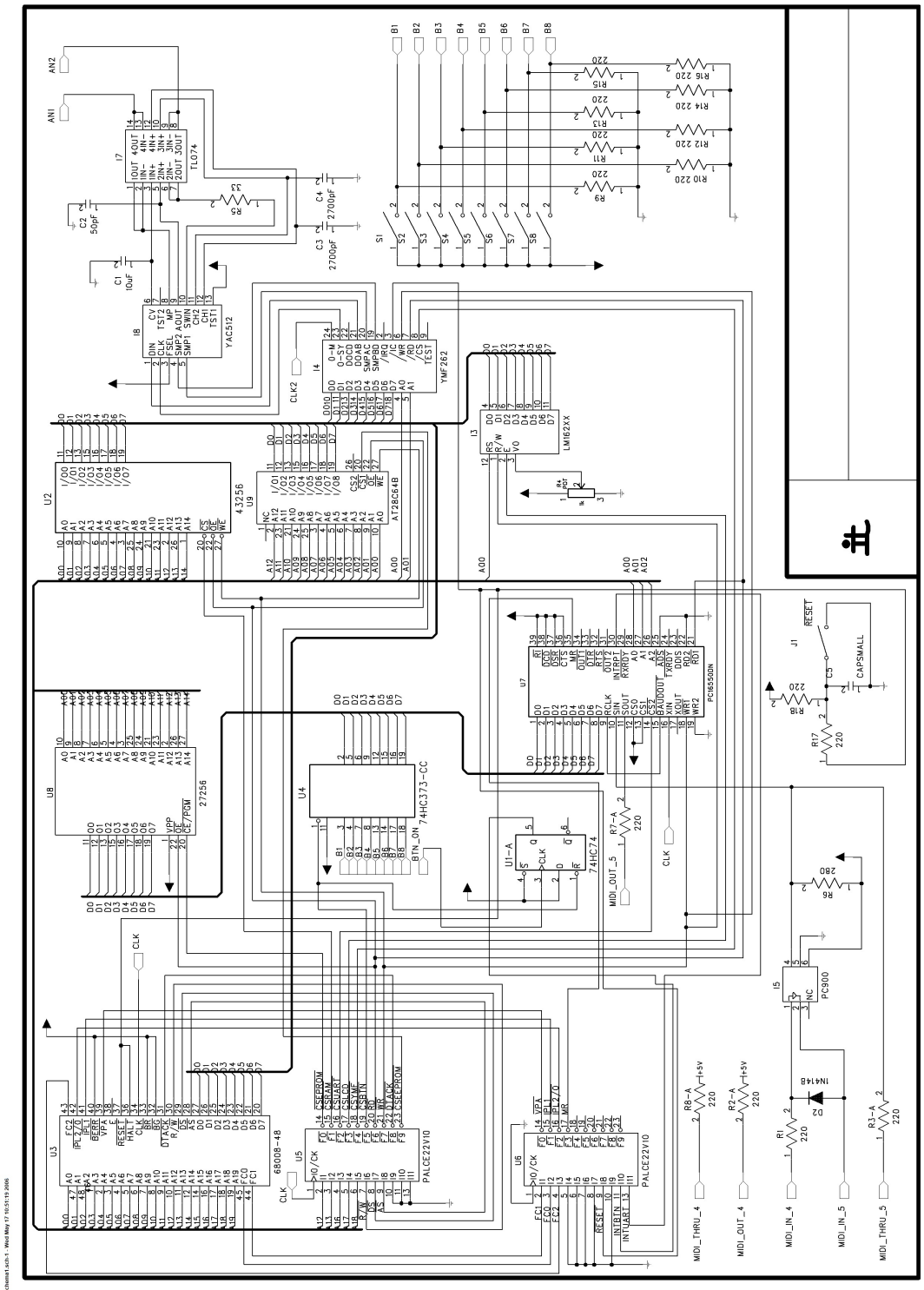
```
FC2 4
AS 5 'Address Strobe från CPU
RESET 9 'Reset (Aktiv låg)
INTBTN 11 'Avbrott från register med strömställare
INTUART 13 'Avbrott från UART
VPA 14 'Autovektoravbrott till CPU
IPL1 15 'Till avbrottsingång IPL1
IPL20 16 'Till avbrottsingång IPL2/0
MR 17 'Reset (Aktiv hög)

start
VPA /= FC0 * FC1 * FC2 * /AS;
IPL20 /= INTUART;
IPL1 /= INTBTN * /INTUART;
MR= /RESET;
end
```

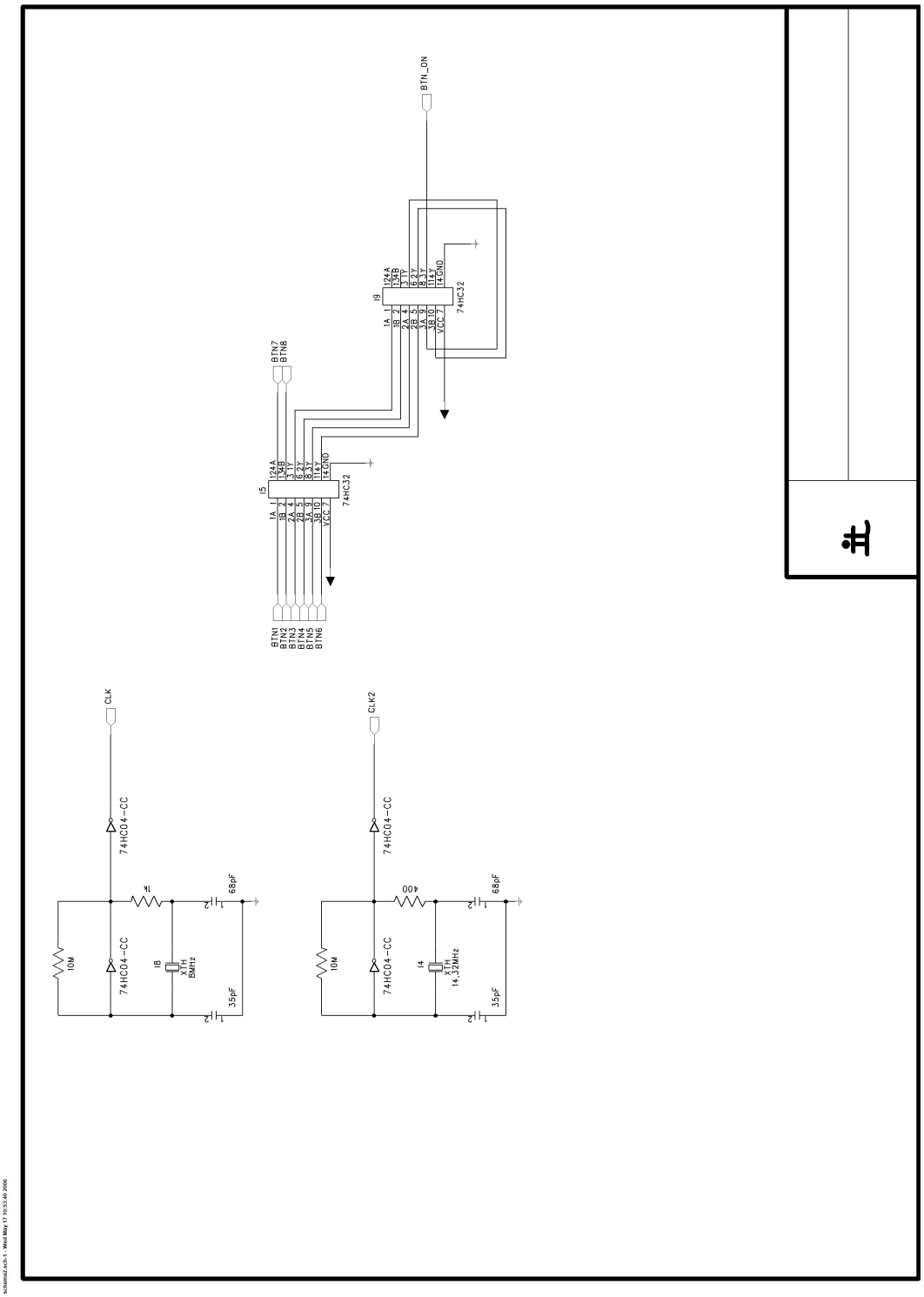
B.2 Addressering

```
Title OPL3Synth
Pattern I/O
Revision 0.0
Author R. Lovblom, G. Ring
Company LTH
Date 02-02-06
device 22v10
CLK 1
A12 2 'Adressbitar
A13 3
A16 4
A17 5
A18 6
RW 7  'Read/Write från CPU
DS 8  'Data Strobe från CPU
AS 9  'Address Strobe från CPU
GND 12
CSEPROM 14 'Chip Select EPROM (Aktiv låg)
CSRAM 15   'Chip Select RAM (Aktiv låg)
CSUART 16  'Chip Select UART (Aktiv låg)
CSLCD 17   'Chip Select LCD (Aktiv hög)
CSYMF 18   'Chip Select YMF262 (Aktiv låg)
CSBTN 19   'Chip Select Register med strömställare (Aktiv låg)
RD 20      'Read enable
WR 21      'Write enable
```

```
DTACK 22   'Data Transfer Acknowledgement till CPU
CSEEPROM 23  'Chip Select EPROM (Aktiv låg)
VCC 24
start
CSEPROM /= /A18 * /A17 * /A16 * /AS;
CSEEPROM /= /A18 * /A17 * A16 * /AS;
CSRAM /= /A18 * A17 * A16 * /AS;
CSUART /= A18 * A17 * /A16 * /A13 * /A12 * /AS;
CSLCD = A18 * A17 * /A16 * /A13 * A12 * /DS;
CSYMF /= A18 * A17 * /A16 * A13 * /A12 * /AS;
CSBTN /= A18 * A17 * /A16 * A13 * A12 * /AS;
RD /= /DS * RW;
WR /= /DS */RW;
DTACK /= /CSEPROM + /CSEEPROM + /CSRAM + /CSUART + CSLCD + /CSYMF + /CSBTN;
end
```

C Kretsschema




BTN_ON
I9
74HC32
BTN7
BTN8
I5
74HC32
BTN1
BTN2
BTN3
BTN4
BTN5
BTN6
CLK
74HC04-CC
1K
68pF
10M
74HC04-CC
I8
XTH
8MHz
35pF
CLK2
74HC04-CC
400
68pF
10M
74HC04-CC
I4
XTH
14,32MHz
35pF