Pratande temometer

Gruppmedlemmar: Marcus Jonsson Ewerbring & Nuo Chen
Email adress: marcusew@kth.se, nuoc@kth.se

Innehållsförteckning:

Produktbeskrivning

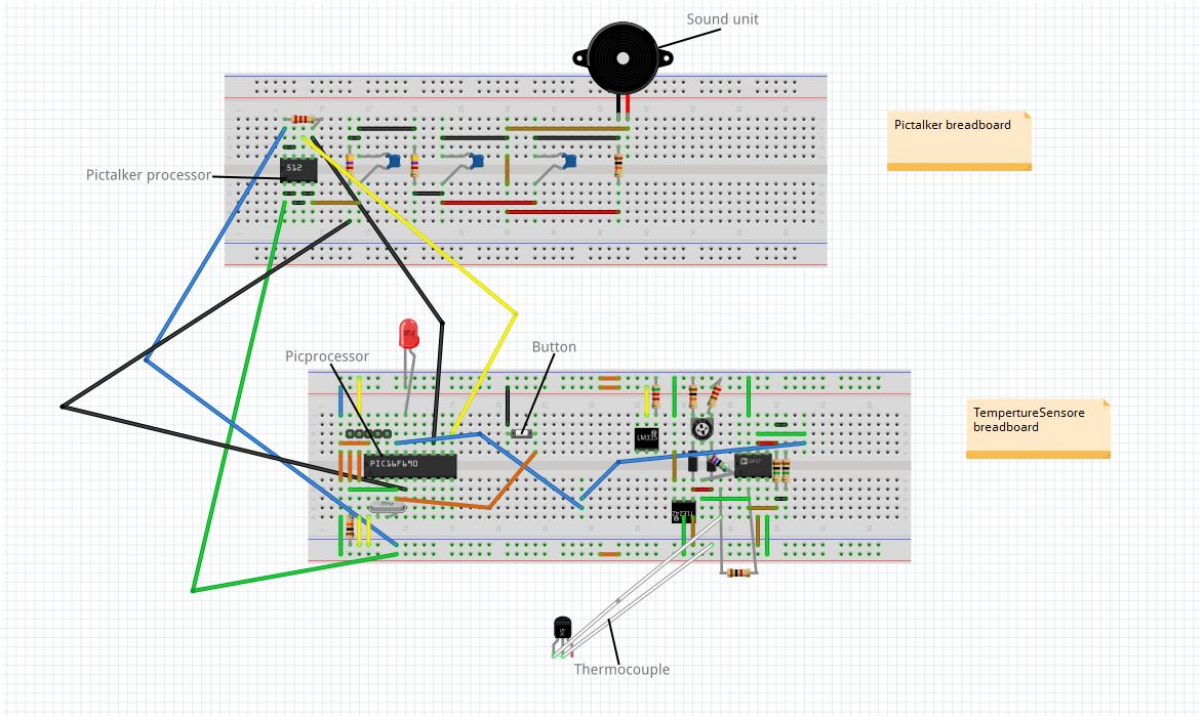

Bild1: Sammanställnings bild på prototypen, obs att termometer sensorn är illustrativ då vi inte använder en sådan i verkligheten.

Prototypen “Pratande termometer” är en termometer som säger vad temperaturen är när knappen på kretsen trycks ner.
Produkten ska bestå av en termometer, en pictalker och en knapp. När knappen trycks in så ska temperaturen mätas och sedan sägas med en röst. Det ska finnas ett test som säger alla meningar som produkten använder, detta för att användarna ska få en uppfattning av hur programmet uttalar de olika orden.

Om vår prototyp skulle användas i en produkt så skulle det vara i en billig termometer som kan säga temperaturen vid ett knapptryck,då hela prototypen är gjord av billigare komponenter. Pictalkern är från 1970 talet så dess komponenter är billigare och drar ner kostnaden för hela produkten.
Prototypen skulle kunna användas i större och mer sofistikerade system men då skulle komponenterna behövas uppdateras till bättre standard då pictalkern generera en röst som är mekanisk. Exempel på större system som skulle kunna använda sig av det här systemet är bilar,ugnar eller andra system som är i behov av att temperaturen ska övervakas och ha möjligheten att ge informationen via röst. Programmet skulle kunna modifieras så att produkten signalerar när temperaturen har överstigit en specifik temperatur men detta är utanför vår projektram.

Följande krav hade vi på vår prototyp:

- Prototypen ska kunna ta in mätvärden från den inkopplade termometern.
- Prototypen ska kunna säga vilken temperatur det är om en knapp trycks ned.

Restriktioner:

Pictalkern kommer inte få stöd för negativa temperaturer och detta för att vi ska kunna hålla oss inom tidsramen av projektet.

Funktionsbeskrivning

Funktionsprototypen består av två kopplingsbord, ett för pictalker enheten och ett för temperaturenheten.

Temperature enheten:

Koplingsbordet med termocouple kopplad till sig representerar termometer kretsen.
Termometer kretsen använder sig av två sensorer som består av metalltrådar av kända material för att bestämma temperaturen på ett objekt eller i ett rum. Temperaturen på den ena metalltråd (sensor) används som referens, och den ska vara känd. Den andra metalltråden används för mätningar. Temperaturskillnaden kommer att ge upphov till en spänning, med vilken kan vi räkna ut temperaturskillnad gentemot referenstemperaturen, och därmed temperaturen på objektet eller i rummet.

Pictalker enheten:

Pictalker en uppsättning av en specifik krets med Piezo Speaker inkopplad och PMW-frekvens för 58 olika toner förprogrammerat i EEPROM-minnet.
Kopplingsbordet som har en högtalare (Piezo Speaker) kopplad till sig är kretsen som genererar ljud till högtalaren, denna krets representerar ljudkortet som skulle sitta i termometern.

Picprossesorn läser av spänningen som den får från termometer kretsen när knappen på temperatur enhetens kopplingsbord trycks in. Spänningen konverteras till ett digitalt värde av picprocessorns ad-omvandlare, som sedan omvandlas till ett ord. Omvandlingen sker genom att för varje siffra i det digitala temperatur värdet sparas dess motsvarande ljudsekvens i en meddelande-array, som sedan läses av i interrupt-rutinen och uppdaterar PWM frekvensen för att skapa de olika toner. På grund av tidsbrist har vi valt att enbart skapa ljudsekvens för 0 till 9, och kombinera dessa för att få 2-siffriga tal. Detta betyder alltså istället för “twenty” så kommer det att sägas “two zero”.
Händelseförloppet beskrivs bild 1 och bild 2 som finns här nedan.

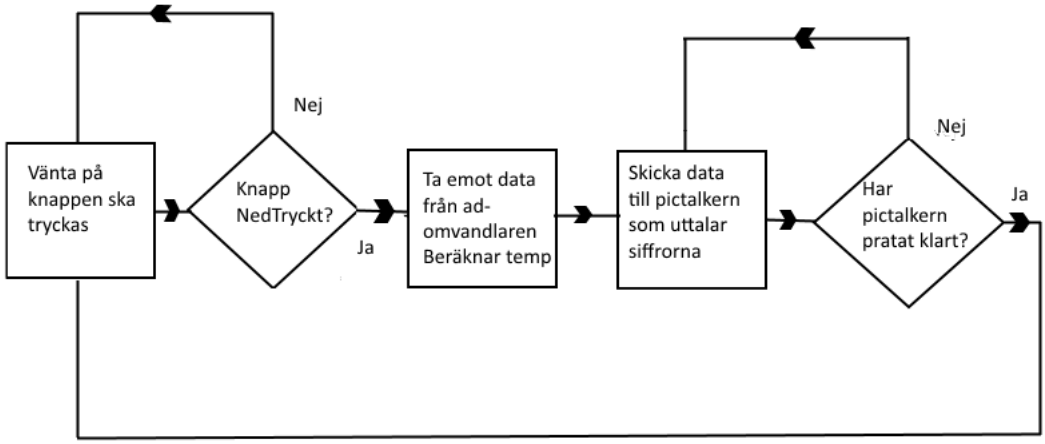

Bild 2: Blockschema som beskriver programmets händelseförlopp.

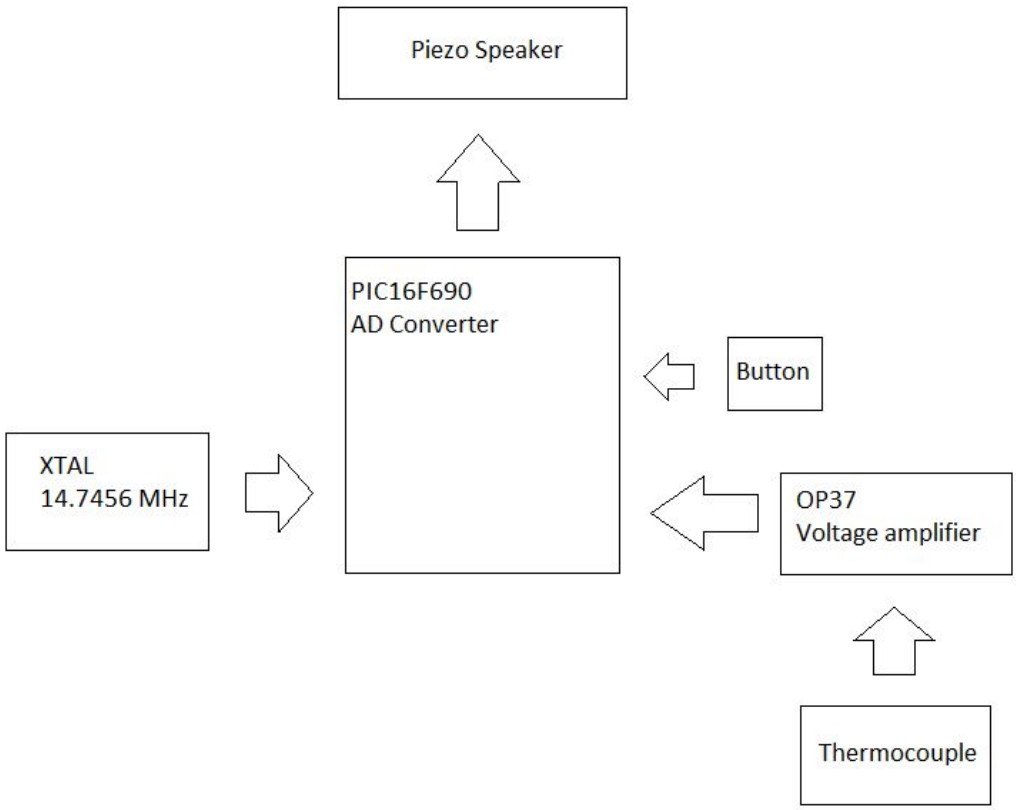

Bild 3: Blockdiagram över hela systemet.

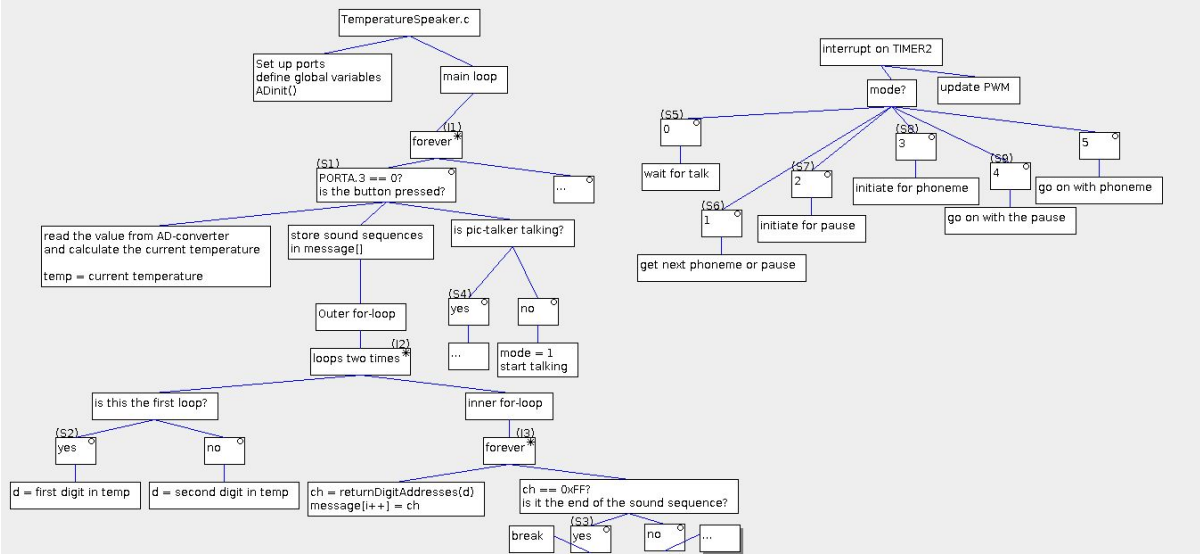

Bild 4: Strukturschema med programmets händelseförlopp

Funktionstest

Följande test utfördes på prototypen.

- Test av pictalker.
- est av termometerkrets.
- Test av mätvärden vid ihopTkoppling av termometer och pictalker.

Vi började att kontrollera att kopplingen var korrekt genom att köra pictalkerns exempelkod, det gjorde vi tills prototypen kunde säga de önskade meningarna. Termometer kretsen testades genom att kolla ifall det blev olika spänningar över kretsen vid olika temperaturer.

När pictalkern och termometerkretsen kopplades ihop så kontrollerades prototypen genom att hålla termometerns referenssensor mot is och mät-sensorn i 20 gradig varm luft. Genom denna metod kunde vi få reda på hur mycket spänningen varierade mellan de olika temperaturerna och räkna ut vilken spänning som ska vara vår referensspänning. Referensspänningen fördes in i prototypens kod som använder den för att mäta temperaturen. Prototypen testades sedan genom att temperatursensorn värmdes upp mellan fingrarna och kyldes ner av en iskub, detta gjordes 10 gånger för att få reda på vår felmarginal. Felmarginalen blev +- 3 grader efter ett flertal försök vilket vi ansåg var acceptabelt.

Fritzing:

Det konstruerades en modell av prototypen i fritzing detta för att produkten ska gå och tillverka.Här nedan finns bilder av prototypen som har blivit konstruerad i Fritzing. Utöver PCB och ritningarna till prototypen så konstruerades ett testprogram *test.c* som testar kretskortets funktioner. Programmet testar först ifall ad-omvandlaren fungerar, programmet signalerar att den funkar genom att lampan på kretskortet blinkar tre gånger. Efter att det testet har genomförts så körs en testfunktion som gör att pictalker kommer räkna från 0 till 9.

Vi valde att pictalkern skulle räkna från 0 till 9 då det är dessa siffror som prototypen använder sig av och det ger användarna en möjlighet att vänja sig med pictalkerns röst då den är svår att tyda. Test programmets kod finns på sida 9.

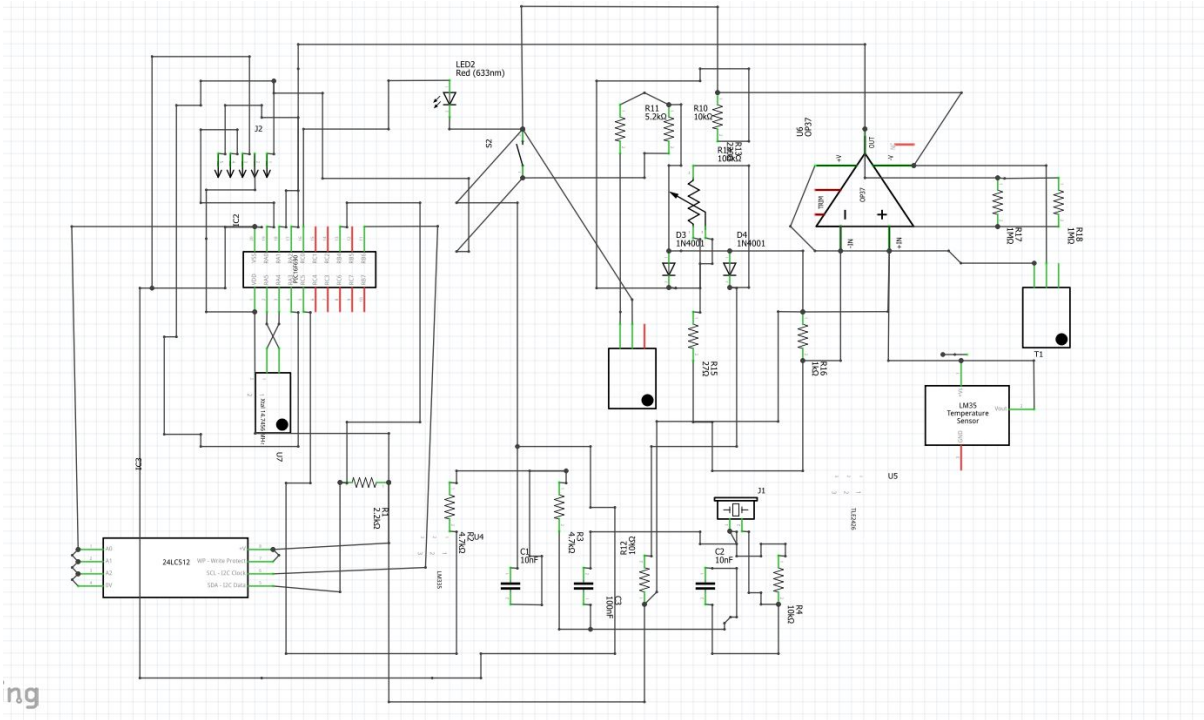

Bild 5: Schema över kretsen.

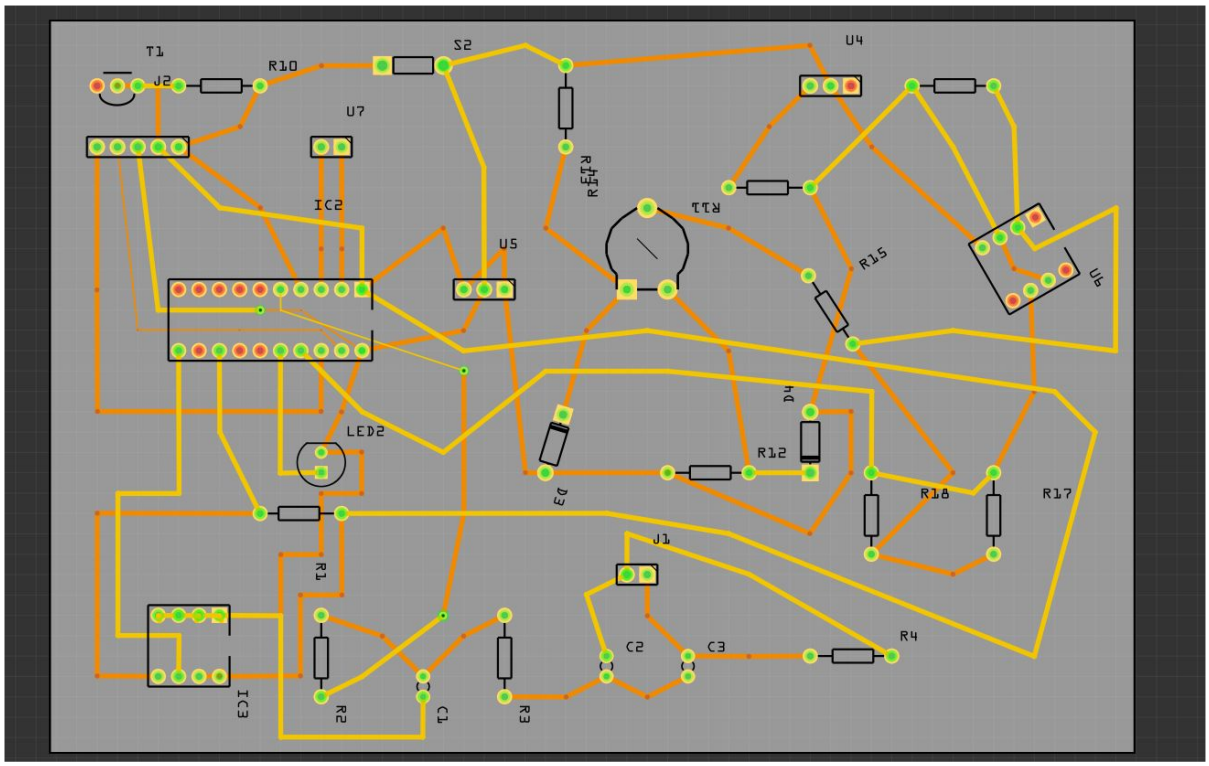

Bild 6: PCB över kretsen, obs temperatursensorn är illustrativ då vi använder 2 metalltrådar som sensorer.

Publicerad kod och länkar till online documentation:

Publicerad kod:

TemperatureSpeaker.c är en modifierad sammansättning av *ADvolt.c* och *pictalk690.c*, båda finns på kurswebben. Funktionerna *returnDigitAddresses(), one(), two(), three(), four(), five(), six(), seven(), eight()* och *nine()* har vi skrivit själva, och större del av “Main loop”. Vi använder oss av B Knudsen Cc5x C-compiler för att kompilera koden.

TemperatureSpeaker.c och fritzing sketch för vår prototyp finns i vår github repository.
Nedan är länken till repository:

https://gits-15.sys.kth.se/nuoc/IE1206_Programmeringsuppgift

Nedan finns koden för *TemperatureSpeaker.c:*

```
/* TemperatureSpeaker.c */
/* This is a modified composition of ADVolt.c and pictalk690.c */
/* Marcus Jonsson Ewerbring & Nuo Chen */
/* B Knudsen Cc5x C-compiler */
/* The following functions have been written by Nuo and Marcus*/ /*returnDigitAddresses(),
one(), two(), three(), four(), five(), six(), seven(),*/ /*eight() and nine() and we have
written the bigger part of "Main loop".*/

#include "16F690.h"
#include "int16Cxx.h"
#pragma config |= 0x00D2  /* use HS-XTAL 14.7456 MHz */
#pragma bit scl_IIC @ PORTB.6
#pragma bit sda_IIC @ PORTB.4

#define  WRITE_sda() TRISB.4 = 0 // SDA must be output when writing
#define  READ_sda()  TRISB.4 = 1 // SDA must be input when reading

#define SCALE_FACTOR 49
#define DECIMALS 4
#define UN_SIGNED 1

#define REF_TEMP 20 //Reference temperature, measured in advance
#define REF_VOLT 25000 //Reference voltage, measured in advance
#pragma codepage 0

/* Function prototypes */
void ADinit( void );
void delay10( char );

void ack_polling(void);
void start(void);
void stop(void);
char read_byte(void);
void send_byte(char);
char read_eeprom(char, char);
void write_eeprom(char, char, char);
char phonemestart_hi(char);
char phonemestart_lo(char);


/* These functions are written by ourselves */
char returnDigitAddress(int, char);
char zero(char);
char one(char);
char two(char);
char three(char);
char four(char);
char five(char);
char six(char);
char seven(char);
char eight(char);
char nine(char);
/* ---------------------------------------*/

/* Global variables */
#pragma rambank 1
char message[61];  // must be long enough for the message
char mode;
char index;
```

```
char phoneme;
unsigned long count;

#pragma origin 4
/* interrupt routine                       */
/* TIMER2 is servoupdating PWM at 7200 Hz */
interrupt int_server( void )
{
  int_save_registers
  char sv_FSR = FSR;       // save FSR
  char sv_PCLATH = PCLATH; // save PCLATH
  /* mode=0 => wait, ready to talk        */
  /* mode=1 => get next phoneme or pause  */
  /* mode=2 => initiate for pause         */
  /* mode=3 => initiate for phoneme       */
  /* mode=4 => go on with the pause       */
  /* mode=5 => go on with the phoneme     */

  char pwm_byte, address_hi, address_lo;
  pwm_byte = 128; // idle value

  switch(mode)
    {
       case 0: // wait for talk
        {
          index = 0;
          break;
        }

       case 1: // get new phoneme or pause
        phoneme = message[index];
       message[index] = 0xFF; // to clean up the array
        index++;  // phoneme is read, so update the index

        if(phoneme == 0xFF) // 0xFF means end of message
          {
            index = 0;
            mode = 0;
            break;
          }

        if(phoneme < 5) // means it is a pause
           {
             mode = 2;
             break;
           }

        /* so it must be a phoneme  */
             mode = 3;
             break;

       case 2: // initiate pause
        if(phoneme == 0) count = 72;   //  10 ms
        if(phoneme == 1) count = 216;  //  30 ms
        if(phoneme == 2) count = 360;  //  50 ms
        if(phoneme == 3) count = 720;  // 100 ms
        if(phoneme == 4) count = 1440; // 200 ms
        mode = 4; // next time, go on with the pause
        break;

       case 3: // initiate phoneme
        /* Random address read */
```

```
        address_hi = phonemestart_hi(phoneme-5);
        address_lo = phonemestart_lo(phoneme-5);
        pwm_byte = read_eeprom(address_hi, address_lo);
        mode = 5; // next time, go on with the phoneme
        break;

       case 4: //  go on with pause
        count--;
        if(count==0) mode = 1; // next time new phoneme/pause
        break;

       case 5: // go on with phoneme
        /* Current adress read, automatic increment */
        start();
        send_byte(0xA1);
        pwm_byte = read_byte(); // phoneme data
        stop();
        if(pwm_byte == 0) // means end of phoneme data
         {
           mode = 1; // next time new phoneme/pause
           pwm_byte = 128;
         }
        break;
    }

  /* update PWM  */
  CCP1CON.4 = pwm_byte.0;
  CCP1CON.5 = pwm_byte.1;
  pwm_byte /= 4;
  CCPR1L = pwm_byte;

  /* return from the interrupt */
  TMR2IF = 0;
  PCLATH = sv_PCLATH;    // restore PCLATH
  FSR = sv_FSR;          // restore FSR
  int_restore_registers
}

void main(void)
{
  unsigned long advalue;
  char i;

   /* Setting up ports */
  TRISC.0 = 0; // lightdiode at RC0 is output
  PORTC.0 = 0; // no light

  ANSELH  = 0 ; // PORTB digital
  PORTB.4 = 0 ; // PORTB.4 pending "0"
  TRISB.4 = 1 ; // simulated open collector
  TRISB.6 = 0 ; // SCL output
  TRISC.5 = 0 ; // CCP PWM output
  TRISA.3 = 1 ; // SW1 input

  ADinit();
  delay10(100);
  /* Setup TIMER2 */
  /*
    0.xxxx.x.xx  - unimplemented
    x.1000.x.xx  Postscaler 8
    x.xxxx.1.xx  TMR2 is on
    x.xxxx.x.00  Prescaler 1
```

```
  */
  T2CON = 0b0.1000.1.00;

  /* Setup CCP1 PWM-mode  */
  /*
    00.xx.xxxx  Single PWM output
    xx.00.xxxx  PWM Duty Two LSB
    xx.xx.1100  11xx when Single PWM-mode
  */
  CCP1CON = 0b00.00.1100 ;

  PR2 = 0x3F; // gives the highest PWM frequency for 8 bit resolution
  mode = 0;   // no talking yet
  TMR2IE  = 1;   /* local enable  */
  PEIE    = 1;   /* peripherals enable */
  GIE     = 1;   /* global enable */
  index =0;

  /* Main loop */
  while(1) {
    if(mode !=0) PORTC.0 = 1; // LED on when talking
    else PORTC.0 = 0;
    while(PORTA.3) ; // wait for key pressed - new measurement
    delay10(100);

    /* Now measure the Voltage [V]  */
    GO=1;         // start AD
    while(GO);    // wait for done
    advalue = ADRESH*256;    // read result 10 bit
    advalue += ADRESL;
    // 1024 -> 5,0000 [V]
    // multiply with integer scalefactor
    advalue *= SCALE_FACTOR;

    signed long temp = (signed long) (advalue - REF_VOLT); // Current temperature =
(voltage difference / K) + reference temperature
    temp = temp / (unsigned) 200; // K = 20.5 mV/C
       //temp += REF_TEMP; // remove this line if measuring with ice as reference
       // temp now holds the current temperature value

    char ch;
    int j, o, d; // temporary variables
       /* Outer for-loop, loops two times */
    for (j = 0; j < 2; j++){
       if (j == 0){
              d = temp / (unsigned) 10; // d = first digit of the temperature value
              i = 0;
              if (d == 0) continue;
        } // Continues to next loop if the first digit is 0
        else d = temp % (unsigned) 10; // d = second digit
        for(o = 0;;i++)
        {
              ch = returnDigitAddress(d, o); // getting the address for phonemes
              message[i]= ch; // and storing them in message[]
              o++;
            if(ch==0xFF)break;
        }
       }

    if (mode == 0) mode = 1; // start talking
    delay10(100);         // Debounce
    PORTC.0=0;          // LED off measurement done
```

```
    while (!PORTA.3) ;  // wait for key released
    delay10(100);         // Debounce
       }
}
```

```
/* ********************************** */
/*     FUNCTIONS FOR ADCONVERTER       */
/* ********************************** */


/* **** ADconverter function ************** */
void ADinit( void )
{
  // AD setup
  ANSEL.2 = 1; // RA2 AN2 analog configurated
  TRISA.2=1;   // AN2 input

  ADCON1 = 0b0.101.0000; // AD conversion clock 'fosc/16'

  /*
     1.x.xxxx.x.x  ADRESH:ADRESL is 10 bit right justified
     x.0.xxxx.x.x  Vref is Vdd
     x.x.0010.x.x  Channel (AN2)
     x.x.xxxx.0.x  Go/!Done - start from program later
     x.x.xxxx.x.1  Enable AD-converter
  */
  ADCON0 = 0b1.0.0010.0.1;
}

/* **** delay function **** */

void delay10( char n)
{
    char i;
    OPTION = 7;
    do  {  i = TMR0 + 39; /* 256 microsec * 39 = 10 ms */
        while ( i != TMR0) ;
    } while ( --n > 0);
}

/* ********************************************** */
/*              FUNCTIONS FOR PICTALKER           */
/* ********************************************** */

/* opcodes are for the chip SP256                */
/* the code "0xFF" is used as "end of sentence"  */
#pragma codepage 2

/* this function contains a switch-statement and returns addresses of phonemes for the
corresponding digit */
char returnDigitAddress(int value, char i) {
  switch (value){
    case 0: return zero(i); break;
    case 1: return one(i); break;
    case 2: return two(i); break;
    case 3: return three(i); break;
```

```
    case 4: return four(i); break;
    case 5: return five(i); break;
    case 6: return six(i); break;
    case 7: return seven(i); break;
    case 8: return eight(i); break;
    case 9: return nine(i); break;

  }
  return 0xFF;
}

/* The following functions contain sound sequences for digit 1 - 9 in Enligsh */
char zero (char i){
  skip(i);
  return 0x37;
  return 0x13;
  return 0x0E;
  return 0x17;
  return 0x04;
  return 0xFF;
}
char one (char i){
  skip(i);
  return 0x2E;
  return 0x20;
  return 0x1B;
  return 0x04;
  return 0xFF;
}
char two (char i){
  skip(i);
  return 0x0D;
  return 0x16;
  return 0x16;
  return 0x04;
  return 0xFF;
}
char three (char i){
  skip(i);
  return 0x1D;
  return 0x0E;
  return 0x13;
  return 0x04;
  return 0xFF;
}
char four (char i){
  skip(i);
  return 0x28;
  return 0x3A;
  return 0x04;
  return 0xFF;
}
char five (char i){
  skip(i);
  return 0x28;
  return 0x06;
  return 0x23;
  return 0x04;
  return 0xFF;
}
char six (char i){
  skip(i);
```

```
  return 0x37;
  return 0x13;
  return 0x29;
  return 0x37;
  return 0x04;
  return 0xFF;
}
char seven (char i){
  skip(i);
  return 0x37;
  return 0x13;
  return 0x23;
  return 0x13;
  return 0x1D;
  return 0x04;
  return 0xFF;
}
char eight (char i){
  skip(i);
  return 0x14;
  return 0x0D;
  return 0x04;
  return 0xFF;
}
char nine (char i){
  skip(i);
  return 0x38;
  return 0x06;
  return 0x1D;
  return 0xFF;
}



/*
    I2C interface for 24LC512
    Nathan Seidle
    nathan.seidle@colorado.edu

    minor changes by William Sandqvist
    to work with the pictalk.c program
    william@kth.se
*/

/* The extra nop()'s are needed if compiled with optimization */

//=================
//Make sure the bits agree with the TRISB statements

//=================
void ack_polling(void)
{
    //wait for chip to respond
    //cycles 17 times to complete
    //write cycle at 4MHz
    while(sda_IIC != 0)
    {
        start();
        send_byte(0xA0);
    }
    stop();
}
```

```
void write_eeprom(char address_hi, char address_lo, char data)
{
    start();
    send_byte(0xA0);
    send_byte(address_hi);
    send_byte(address_lo);
    send_byte(data);
    stop();
}

char read_eeprom(char address_hi, char address_lo)
{

    char byte;

    start();
    send_byte(0xA0); // control byte = write
    send_byte(address_hi);
    send_byte(address_lo);
    stop();

    start();
    send_byte(0xA1); // control byte = read
    byte = read_byte();
    stop();

    return byte;
}

void start(void)
{
    WRITE_sda();
    sda_IIC = 0;
}

void stop(void)
{
    scl_IIC = 0;

    WRITE_sda();
    sda_IIC = 0;
    // nop();
    // nop();
    nop();
    nop();
    scl_IIC = 1;
    // nop();
    // nop();
    nop();
    sda_IIC = 1;
}

char read_byte(void)
{
    char j, in_byte;

    scl_IIC = 0;

    READ_sda();

    for(j = 0 ; j < 8 ; j++)
```

```
    {
        scl_IIC = 0;
        // nop();
        // nop();
        // nop();
        nop();
        scl_IIC = 1;

        in_byte = rl(in_byte);
        in_byte.0 = sda_IIC;
    }

    return in_byte;
}

void send_byte(char out_byte)
{
    char i;

    WRITE_sda();

    for( i = 0 ; i < 8 ; i++ )
    {
        out_byte = rl(out_byte);
        scl_IIC = 0;
        sda_IIC = Carry;
        scl_IIC = 1;
        nop();
    }

    //read ack.
    scl_IIC = 0;
    READ_sda();
    scl_IIC = 1;
}

/* lookup tables for allphone start adresses in the I2C-EEPROM        */
/* all allphone data ends with "0" in the I2C-EEPROM                  */
/* pauses 0x00, 0x01, 0x02, 0x03, 0x04 are not stored in EEPROM       */
/* so opcodes for SP256 with the offset of -5 should be used as index */

char phonemestart_hi(char index)
{
  skip(index);
  return 0x00; // 0x05  OY  as in vOIce
  return 0x09; // 0x06  AY  as in kIte
  return 0x0E; // 0x07  EH  as in End
  return 0x10; // 0x08  KK3 as in Crane
  return 0x13; // 0x09  PP  as in Pow
  return 0x18; // 0x0A  JH  as in JudGe
  return 0x1B; // 0x0B  NN1 as in thiN
  return 0x21; // 0x0C  IH  as in sIt
  return 0x23; // 0x0D  TT2 as in To
  return 0x26; // 0x0E  RR1 as in Read
  return 0x2B; // 0x0F  AX  as in sUceed
  return 0x2D; // 0x10  MM  as in ilk
  return 0x33; // 0x11  TT1 as in parTs
  return 0x36; // 0x12  DH1 as in THis
  return 0x3A; // 0x13  IY  as in sEE
  return 0x40; // 0x14  EY  as in trAY
  return 0x46; // 0x15  DD1 as in enD
  return 0x48; // 0x16  UW1 as in tO
```

```
  return 0x4B; // 0x17  AO  as in AUght
  return 0x4E; // 0x18  AA  as in hOt
  return 0x51; // 0x19  YY2 as in Yes
  return 0x56; // 0x1A  AE  as in hAt
  return 0x59; // 0x1B  HH1 as in He
  return 0x5C; // 0x1C  BB1 as in riB
  return 0x5E; // 0x1D  TH  as in THin
  return 0x62; // 0x1E  UH  as in bOOk
  return 0x65; // 0x1F  UW2 as in fOOd
  return 0x6A; // 0x20  AW  as in dOWn
  return 0x72; // 0x21  DD2 as in Down
  return 0x74; // 0x22  GG3 as in peG
  return 0x78; // 0x23  VV  as in Vest
  return 0x7C; // 0x24  GG1 as in Guest
  return 0x7E; // 0x25  SH  as in SHip
  return 0x85; // 0x26  ZH  as in pleaSure
  return 0x89; // 0x27  RR2 as in cRane
  return 0x8D; // 0x28  FF  as in Food
  return 0x90; // 0x29  KK2 as in speaK
  return 0x95; // 0x2A  KK1 as in Can't
  return 0x99; // 0x2B  ZZ  as in Zoo
  return 0x9D; // 0x2C  NG  as in aNGer
  return 0xA4; // 0x2D  LL  as in Like
  return 0xA6; // 0x2E  WW  as in We
  return 0xAB; // 0x2F  XR  as in stARs
  return 0xB2; // 0x30  WH  as in WHig
  return 0xB8; // 0x31  YY1 as in cUte
  return 0xBB; // 0x32  CH  as in CHurCH
  return 0xBF; // 0x33  ER1 as in lettER
  return 0xC3; // 0x34  ER2 as in fERn
  return 0xCA; // 0x35  OW  as in zOne
  return 0xCF; // 0x36  DH2 as in baTHe
  return 0xD4; // 0x37  SS  as in veSt
  return 0xD7; // 0x38  NN2 as in No
  return 0xDB; // 0x39  HH2 as in Hoe
  return 0xDF; // 0x3A  OR  as in fORtune
  return 0xE6; // 0x3B  AR  as in alARm
  return 0xED; // 0x3C  YR  as in hEAr
  return 0xF4; // 0x3D  GG2 as in Got
  return 0xF6; // 0x3E  EL  as in angLE
  return 0xFB; // 0x3F  BB2 as in Beast
}

char phonemestart_lo(char index)
{
  skip(index);
  return 0x06; // 0x05  OY  as in vOIce
  return 0x00; // 0x06  AY  as in kIte
  return 0x80; // 0x07  EH  as in End
  return 0xC0; // 0x08  KK3 as in Crane
  return 0x80; // 0x09  PP  as in Pow
  return 0x40; // 0x0A  JH  as in udGe
  return 0xC0; // 0x0B  NN1 as in thiN
  return 0x40; // 0x0C  IH  as in sIt
  return 0x00; // 0x0D  TT2 as in To
  return 0x40; // 0x0E  RR1 as in Read
  return 0x40; // 0x0F  AX  as in sUceed
  return 0xC0; // 0x10  MM  as in Milk
  return 0x80; // 0x11  TT1 as in arTs
  return 0x40; // 0x12  DH1 as in THis
  return 0xC0; // 0x13  IY  as in sEE
  return 0x40; // 0x14  EY  as in trAY
```

```
  return 0x80; // 0x15  DD1 as in enD
  return 0x40; // 0x16  UW1 as in tO
  return 0x00; // 0x17  AO  as in AUght
  return 0x80; // 0x18  AA  as in hOt
  return 0xC0; // 0x19  YY2 as in Yes
  return 0x40; // 0x1A  AE  as in hAt
  return 0x80; // 0x1B  HH1 as in He
  return 0x80; // 0x1C  BB1 as in riB
  return 0x00; // 0x1D  TH  as in THin
  return 0x40; // 0x1E  UH  as in bOOk
  return 0x00; // 0x1F  UW2 as in fOOd
  return 0x80; // 0x20  AW  as in dOWn
  return 0x40; // 0x21  DD2 as in Down
  return 0xC0; // 0x22  GG3 as in peG
  return 0x40; // 0x23  VV  as in Vest
  return 0x40; // 0x24  GG1 as in Guest
  return 0xC0; // 0x25  SH  as in SHip
  return 0x00; // 0x26  ZH  as in pleaSure
  return 0x80; // 0x27  RR2 as in cRane
  return 0x40; // 0x28  FF  as in Food
  return 0xC0; // 0x29  KK2 as in speaK
  return 0x40; // 0x2A  KK1 as in Can't
  return 0x00; // 0x2B  ZZ  as in Zoo
  return 0xC0; // 0x2C  NG  as in aNGer
  return 0x00; // 0x2D  LL  as in Like
  return 0xC0; // 0x2E  WW  as in We
  return 0x40; // 0x2F  XR  as in stARs
  return 0xC0; // 0x30  WH  as in WHig
  return 0x00; // 0x31  YY1 as in cUte
  return 0x00; // 0x32  CH  as in CHurCH
  return 0xC0; // 0x33  ER1 as in lettER
  return 0x80; // 0x34  ER2 as in fERn
  return 0x00; // 0x35  OW  as in zOne
  return 0x40; // 0x36  DH2 as in baTHe
  return 0xC0; // 0x37  SS  as in veSt
  return 0x00; // 0x38  NN2 as in No
  return 0x80; // 0x39  HH2 as in Hoe
  return 0x80; // 0x3A  OR  as in fORtune
  return 0xC0; // 0x3B  AR  as in alARm
  return 0x00; // 0x3C  YR  as in hEAr
  return 0x80; // 0x3D  GG2 as in Got
  return 0xC0; // 0x3E  EL  as in angLE
  return 0x00; // 0x3F  BB2 as in Beast
}
```

```
/* ********************************** */
/*            HARDWARE                */
/* ********************************** */

/*
   pictalk690.c  PICTalker program for 16F690 on PK2 Starterkit
   combined with ADVolt.c program for temperature measuring

          _____________  _____________
         |             \/             |                       J1          ________
    +5V---|Vdd        16F690       Vss|---GND                 1 RA5 --- |XTAL    |
  XTAL--- |RA5               RA0/(PGD)|bbTx ->- PK2Rx/PGD     2 RA4 --- |14.7456 |
         |RA4/AN3   AN1/REF/RA1/(PGC)|------<- PGC           3 RA3     |________|
         |RA3/!MCLR/(Vpp) RA2/AN2/INT|-<- U (thermometer)    4 RC5 ->- PWM Speach
PWM Speach|RC5/CCP                 RC0|->- LED                5 RC4     LP-filter
      PWM |RC4                     RC1|
         |RC3                     RC2|
         |RC6                     RB4|-<- I2C
         |RC7                  RB5/Rx|
         |RB7/Tx                  RB6|->- I2C
         |___________________________|
*/

/* On mini Breadboard
  I2C EEPROM with allophones - 2k pullup on SDA

        ___  ___
       |   \/   |
 GND---|A0   Vcc|---+5V
 GND---|A1    WP|---+5V
 GND---|A2   SCL|-<-RB6/SCL
 GND---|GND  SDA|->-RB4/SDA
       |________|
  PWM LP-filter. RC-ladder: 4k7, 10n, 4k7, 10n
*/
```