

```

////////////////////////////////////
// CW Decoder made by Hjalmar Skovholm Hansen OZ1JHM  VER 1.01      //
// Feel free to change, copy or what ever you like but respect    //
// that license is http://www.gnu.org/copyleft/gpl.html           //
// Discuss and give great ideas on                                //
// https://groups.yahoo.com/neo/groups/oz1jhm/conversations/messages //
////////////////////////////////////

////////////////////////////////////
// Read more here http://en.wikipedia.org/wiki/Goertzel_algorithm  //
// if you want to know about FFT the http://www.dspguide.com/pdfbook.htm //
////////////////////////////////////

#include <LiquidCrystal.h>

////////////////////////////////////
// select the pins used on the LCD panel                          //
////////////////////////////////////
// LiquidCrystal lcd(RS, E, D4, D5, D6, D7) //
////////////////////////////////////

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

const int columns = 20; /// have to be 16 or 20
const int rows = 4;    /// have to be 2 or 4

int lcdindex = 0;
int line1[columns];
int line2[columns];

////////////////////////////////////
// Define 8 specials letters //
////////////////////////////////////

byte U_umlaut[8] = {B01010,B00000,B10001,B10001,B10001,B10001,B01110,B00000}; //
'Û'
byte O_umlaut[8] = {B01010,B00000,B01110,B10001,B10001,B10001,B01110,B00000}; //
'Ö'
byte A_umlaut[8] = {B01010,B00000,B01110,B10001,B11111,B10001,B10001,B00000}; //
'Ä'
byte AE_capital[8] = {B01111,B10100,B10100,B11110,B10100,B10100,B10111,B00000}; //
'Æ'
byte OE_capital[8] = {B00001,B01110,B10011,B10101,B11001,B01110,B10000,B00000}; //
'Ø'
byte fullblock[8] = {B11111,B11111,B11111,B11111,B11111,B11111,B11111,B11111};
byte AA_capital[8] = {B00100,B00000,B01110,B10001,B11111,B10001,B10001,B00000}; //
'Å'
byte emtyblock[8] = {B00000,B00000,B00000,B00000,B00000,B00000,B00000,B00000};

int audioInPin = A1;
int audioOutPin = 10;
int ledPin = 13;

float magnitude ;
int magnitudelimit = 100;
int magnitudelimit_low = 100;
int realstate = LOW;
int realstatebefore = LOW;
int filteredstate = LOW;
int filteredstatebefore = LOW;

////////////////////////////////////
// The sampling frq will be 8928 on a 16 mhz                      //
// without any prescaler etc                                     //
// because we need the tone in the center of the bins          //
// you can set the tone to 496, 558, 744 or 992                //
// then n the number of samples which give the bandwidth //

```

```

// can be (8928 / tone) * 1 or 2 or 3 or 4 etc //
// init is 8928/558 = 16 *4 = 64 samples //
// try to take n = 96 or 128 ;o) //
// 48 will give you a bandwidth around 186 hz //
// 64 will give you a bandwidth around 140 hz //
// 96 will give you a bandwidth around 94 hz //
// 128 will give you a bandwidth around 70 hz //
// BUT remember that high n take a lot of time //
// so you have to find the compromise - i use 48 //
////////////////////////////////////

float coeff;
float Q1 = 0;
float Q2 = 0;
float sine;
float cosine;
float sampling_freq=8928.0;
float target_freq=558.0; /// adjust for your needs see above
float n=48.0; /// if you change her please change next line also
int testData[48];

////////////////////////////////////
// Noise Blanker time which //
// shall be computed so //
// this is initial //
////////////////////////////////////
int nbtime = 6; /// ms noise blanker

long starttimehigh;
long highduration;
long lasthighduration;
long hightimesavg;
long lowtimesavg;
long startttimelow;
long lowduration;
long laststarttime = 0;

char code[20];
int stop = LOW;
int wpm;

////////////////////////////////////
// init setup //
////////////////////////////////////
void setup() {

////////////////////////////////////
// The basic goertzel calculation //
////////////////////////////////////
int k;
float omega;
k = (int) (0.5 + ((n * target_freq) / sampling_freq));
omega = (2.0 * PI * k) / n;
sine = sin(omega);
cosine = cos(omega);
coeff = 2.0 * cosine;

////////////////////////////////////
// define special characters //
////////////////////////////////////
lcd.createChar(0, U_umlaut); // German
lcd.createChar(1, O_umlaut); // German, Swedish
lcd.createChar(2, A_umlaut); // German, Swedish
lcd.createChar(3, AE_capital); // Danish, Norwegian
lcd.createChar(4, OE_capital); // Danish, Norwegian
lcd.createChar(5, fullblock);
lcd.createChar(6, AA_capital); // Danish, Norwegian, Swedish

```

```

lcd.createChar(7, emptyblock);
lcd.clear();

Serial.begin(115200);
pinMode(ledPin, OUTPUT);
lcd.begin(columns, rows);
for (int index = 0; index < columns; index++){
    line1[index] = 32;
    line2[index] = 32;
}

}

////////////////////
// main loop //
////////////////////
void loop() {

    //////////////////////
    // The basic where we get the tone //
    //////////////////////

    for (char index = 0; index < n; index++)
    {
        testData[index] = analogRead(audioInPin);
    }
    for (char index = 0; index < n; index++){
        float Q0;
        Q0 = coeff * Q1 - Q2 + (float) testData[index];
        Q2 = Q1;
        Q1 = Q0;
    }
    float magnitudeSquared = (Q1*Q1)+(Q2*Q2)-Q1*Q2*coeff; // we do only need the real
part //
    magnitude = sqrt(magnitudeSquared);
    Q2 = 0;
    Q1 = 0;

    //Serial.print(magnitude); Serial.println(); //// here you can measure magnitude
for setup..

    //////////////////////
    // here we will try to set the magnitude limit automatic //
    //////////////////////

    if (magnitude > magnitudelimit_low){
        magnitudelimit = (magnitudelimit +((magnitude - magnitudelimit)/6)); /// moving
average filter
    }

    if (magnitudelimit < magnitudelimit_low)
        magnitudelimit = magnitudelimit_low;

    //////////////////////
    // now we check for the magnitude //
    //////////////////////

    if(magnitude > magnitudelimit*0.6) // just to have some space up
        realstate = HIGH;
    else
        realstate = LOW;

    //////////////////////
    // here we clean up the state with a noise blanker //
    //////////////////////

    if (realstate != realstatebefore){
        laststarttime = millis();

```

```

}
if ((millis()-laststarttime)> nvertime){
    if (realstate != filteredstate){
        filteredstate = realstate;
    }
}

////////////////////////////////////
// Then we do want to have some durations on high and low //
////////////////////////////////////

if (filteredstate != filteredstatebefore){
    if (filteredstate == HIGH){
        starttimehigh = millis();
        lowduration = (millis() - startttimelow);
    }

    if (filteredstate == LOW){
        startttimelow = millis();
        highduration = (millis() - starttimehigh);
        if (highduration < (2*hightimesavg) || hightimesavg == 0){
            hightimesavg = (highduration+hightimesavg+hightimesavg)/3;
// now we know avg dit time ( rolling 3 avg)
        }
        if (highduration > (5*hightimesavg) ){
            hightimesavg = highduration+hightimesavg; // if speed
decrease fast ..
        }
    }
}

////////////////////////////////////
// now we will check which kind of baud we have - dit or dah //
// and what kind of pause we do have 1 - 3 or 7 pause //
// we think that hightimeavg = 1 bit //
////////////////////////////////////

if (filteredstate != filteredstatebefore){
    stop = LOW;
    if (filteredstate == LOW){ //// we did end a HIGH
        if (highduration < (hightimesavg*2) && highduration > (hightimesavg*0.6)){ /// 0.6
filter out false dits
            strcat(code, ".");
            Serial.print(".");
        }
        if (highduration > (hightimesavg*2) && highduration < (hightimesavg*6)){
            strcat(code, "-");
            Serial.print("-");
            wpm = (wpm + (1200/((highduration)/3)))/2; //// the most precise we can do
;o)
        }
    }
}

if (filteredstate == HIGH){ //// we did end a LOW

float lacktime = 1;
if(wpm > 25)lacktime=1.0; /// when high speeds we have to have a little more
pause before new letter or new word
if(wpm > 30)lacktime=1.2;
if(wpm > 35)lacktime=1.5;

if (lowduration > (hightimesavg*(2*lacktime)) && lowduration <
hightimesavg*(5*lacktime)){ // letter space
    docode();
    code[0] = '\0';
    Serial.print("/");
}
if (lowduration >= hightimesavg*(5*lacktime)){ // word space

```

```

    docode();
        code[0] = '\0';
        printascii(32);
        Serial.println();
    }
}
}

////////////////////////////////////
// write if no more letters //
////////////////////////////////////

if ((millis() - startttimelow) > (highduration * 6) && stop == LOW){
    docode();
    code[0] = '\0';
    stop = HIGH;
}

////////////////////////////////////
// we will turn on and off the LED //
// and the speaker //
////////////////////////////////////

if(filteredstate == HIGH){
    digitalWrite(ledPin, HIGH);
    tone(audioOutPin,target_freq);
}
else{
    digitalWrite(ledPin, LOW);
    noTone(audioOutPin);
}

////////////////////////////////////
// the end of main loop clean up//
////////////////////////////////////
updateinfolinelcd();
realstatebefore = realstate;
lasthighduration = highduration;
filteredstatebefore = filteredstate;
}

////////////////////////////////////
// translate cw code to ascii //
////////////////////////////////////

void docode(){
    if (strcmp(code,".-") == 0) printascii(65);
    if (strcmp(code,"-...") == 0) printascii(66);
    if (strcmp(code,"-.-.") == 0) printascii(67);
    if (strcmp(code,"-..") == 0) printascii(68);
    if (strcmp(code,".") == 0) printascii(69);
    if (strcmp(code,"..-") == 0) printascii(70);
    if (strcmp(code,"--.") == 0) printascii(71);
    if (strcmp(code,"...") == 0) printascii(72);
    if (strcmp(code,"..") == 0) printascii(73);
    if (strcmp(code,".---") == 0) printascii(74);
    if (strcmp(code,"-.-") == 0) printascii(75);
    if (strcmp(code,"-..") == 0) printascii(76);
    if (strcmp(code,"--") == 0) printascii(77);
    if (strcmp(code,"-.") == 0) printascii(78);
    if (strcmp(code,"---") == 0) printascii(79);
    if (strcmp(code,"--.") == 0) printascii(80);
    if (strcmp(code,"--.-") == 0) printascii(81);
    if (strcmp(code,".-.") == 0) printascii(82);
    if (strcmp(code,"...") == 0) printascii(83);
    if (strcmp(code,"-") == 0) printascii(84);
    if (strcmp(code,"..-") == 0) printascii(85);
}

```

```

if (strcmp(code,"...-") == 0) printascii(86);
if (strcmp(code,".--") == 0) printascii(87);
if (strcmp(code,"-..") == 0) printascii(88);
if (strcmp(code,"-.-") == 0) printascii(89);
if (strcmp(code,"--..") == 0) printascii(90);

if (strcmp(code,".----") == 0) printascii(49);
if (strcmp(code,"..---") == 0) printascii(50);
if (strcmp(code,"...--") == 0) printascii(51);
if (strcmp(code,"....-") == 0) printascii(52);
if (strcmp(code,".....") == 0) printascii(53);
if (strcmp(code,"-....") == 0) printascii(54);
if (strcmp(code,"--...") == 0) printascii(55);
if (strcmp(code,"---..") == 0) printascii(56);
if (strcmp(code,"----.") == 0) printascii(57);
if (strcmp(code,"-----") == 0) printascii(48);

if (strcmp(code,"..--..") == 0) printascii(63);
if (strcmp(code,".-.-.-") == 0) printascii(46);
if (strcmp(code,"--..--") == 0) printascii(44);
if (strcmp(code,"-.-.-") == 0) printascii(33);
if (strcmp(code,"-.-.-") == 0) printascii(64);
if (strcmp(code,"---...") == 0) printascii(58);
if (strcmp(code,"-....-") == 0) printascii(45);
if (strcmp(code,"-.-.-") == 0) printascii(47);

if (strcmp(code,"-.-.-") == 0) printascii(40);
if (strcmp(code,"-.-.-") == 0) printascii(41);
if (strcmp(code,"-....") == 0) printascii(95);
if (strcmp(code,"...-.-") == 0) printascii(36);
if (strcmp(code,"...-.-") == 0) printascii(62);
if (strcmp(code,".-.-.") == 0) printascii(60);
if (strcmp(code,"...-") == 0) printascii(126);
////////////////////
// The specials //
////////////////////
if (strcmp(code,".-.-") == 0) printascii(3);
if (strcmp(code,"---.") == 0) printascii(4);
if (strcmp(code,".-.-") == 0) printascii(6);

}

////////////////////
// print the ascii code to the lcd //
// one a time so we can generate //
// special letters //
////////////////////
void printascii(int asciinumber){

int fail = 0;
if (rows == 4 and colums == 16)fail = -4; /// to fix the library problem with 4*16
display http://forum.arduino.cc/index.php/topic,14604.0.html

if (lcdindex > colums-1){
  lcdindex = 0;
  if (rows==4){
    for (int i = 0; i <= colums-1 ; i++){
      lcd.setCursor(i,rows-3);
      lcd.write(line2[i]);
      line2[i]=line1[i];
    }
  }
  for (int i = 0; i <= colums-1 ; i++){
    lcd.setCursor(i+fail,rows-2);
    lcd.write(line1[i]);
    lcd.setCursor(i+fail,rows-1);
    lcd.write(32);
  }
}

```

```

}
line1[lcdindex]=asciinumbe;
lcd.setCursor(lcdindex+fail,rows-1);
lcd.write(asciinumbe);
lcdindex += 1;
}

void updateinfoLineLCD() {
////////////////////////////////////
// here we update the upper line //
// with the speed. //
////////////////////////////////////

int place;
if (rows == 4) {
    place = columns/2;
} else {
    place = 2;
}

    if (wpm<10) {
        lcd.setCursor((place)-2,0);
        lcd.print("0");
        lcd.setCursor((place)-1,0);
        lcd.print(wpm);
        lcd.setCursor((place),0);
        lcd.print(" WPM");
    }
    else {
        lcd.setCursor((place)-2,0);
        lcd.print(wpm);
        lcd.setCursor((place),0);
        lcd.print(" WPM ");
    }
}
}

```