Automatic Pilot Interaction System (APIS)

Autor: Mark Juhrig

Datum: 24.04.2023

Revision: E

Änderungen:

Rev. A: Erstausgabe
Rev. B: Modellflug- & KFZ-Ansagen zugefügt
Rev. C: Timing für PCL-Klick optimiert, Watchdog (8 Sekunden)
Rev. D: PCL-Optimierung
Rev. E: radi.mp3 anstelle info.mp3 (… Radio …), PCL-Klickzeit: 1000 ms, Windsensor Winkel-Offset

Inhalt

Komponenten ... 3
Thies Wetter Sensor ... 3
Arduino Uno ... 4
Peripherie: ... 4
Aufbau ... 9
Steckreihenfolge ... 9
Zuordnung der Arduino Ein-/Ausgänge ... 9
Eingänge ... 9
Ausgänge ... 9
Audio/MP3-Shield ... 10
Relays shield ... 10
Verdrahtung (Arduino) ... 10
Verdrahtung (Becker AR6201) ... 11
Stromaufnahme ... 11
Software (Arduino Sketch) ... 12
Quellcode ... 12
Arduino Bibliotheken ... 21
MP3-Shield Modifikation ... 21
MP3-Dateien ... 24
Ansage Format ... 25
Benutzerhandbuch ... 26
Funktionen ... 26
Display ... 27
Einstellung Funkgerät ... 28

Squelch-Einstellung 28
Mikrofonempfindlichkeit..... 28

Komponenten

Thies Wetter Sensor

Typ:	Clima Sensor US TFB
Artikelnummer:	4.9201.00.000
Schnittstelle:	RS485, 9600 Baud
Datentelegram:	
Frequenz:	1 Hz
Daten:	Windspeed (m/s), Richtung, Temperatur, Luftfeutigkeit, Luftdruck, Datum, Uhrzeit, Prüfziffer
Beispiel:	001.4 272 +03.3 94 985.0 23.03.21 06:59:46 *27 siehe Telegramm VDTHP in 4.920x.x0.xxx_ClimaSensor_US_d.pdf und TR (Telegramm Request)

Arduino Uno

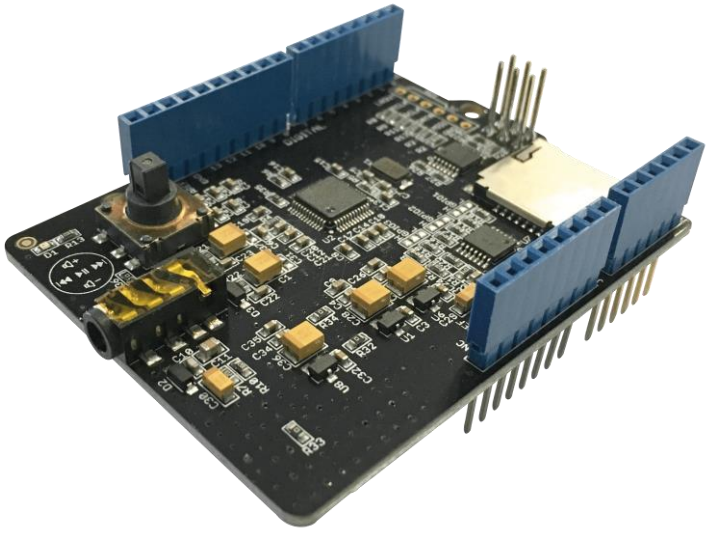
Arduino Uno R3

https://www.reichelt.de/arduino-uno-rev-3-smd-variante-atmega328-usb-arduino-uno-p119045.html?search=arduino+uno

Peripherie:

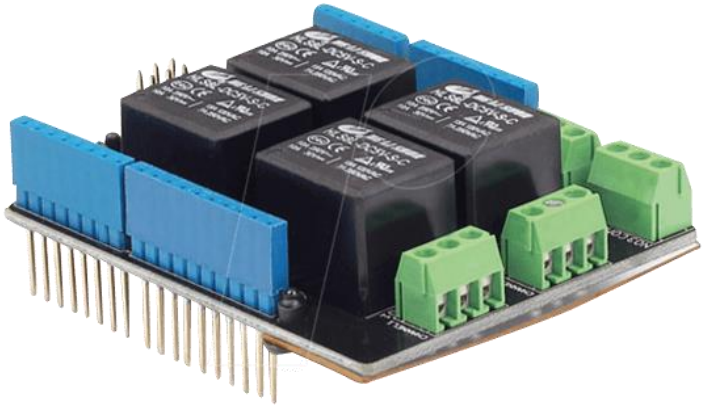
Stimmausgabe: Arduino Music Shield V2.2 + Micro SD-Karte (max. 2 GByte)

https://www.reichelt.de/arduino-shield-musik-v2-0-vs1053b-ard-shd-music-v2-p191283.html?PROVID=2788



PPT+PCL-Ausgang: Arduino Relay Shield V3.0

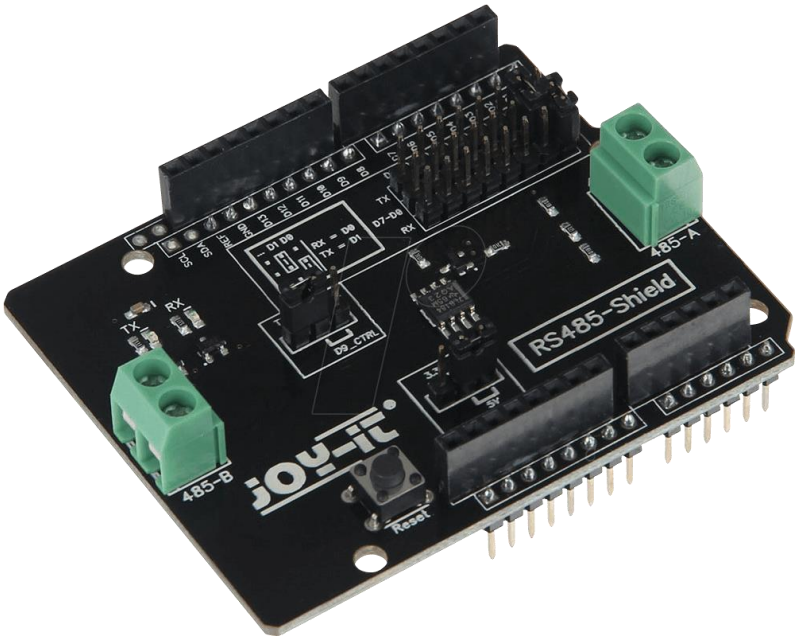
https://www.reichelt.de/arduino-shield-relais-v3-hls8l-dc5v-s-c-ard-shd-relay-v3-p191269.html?&trstct=pos_4&nbc=1

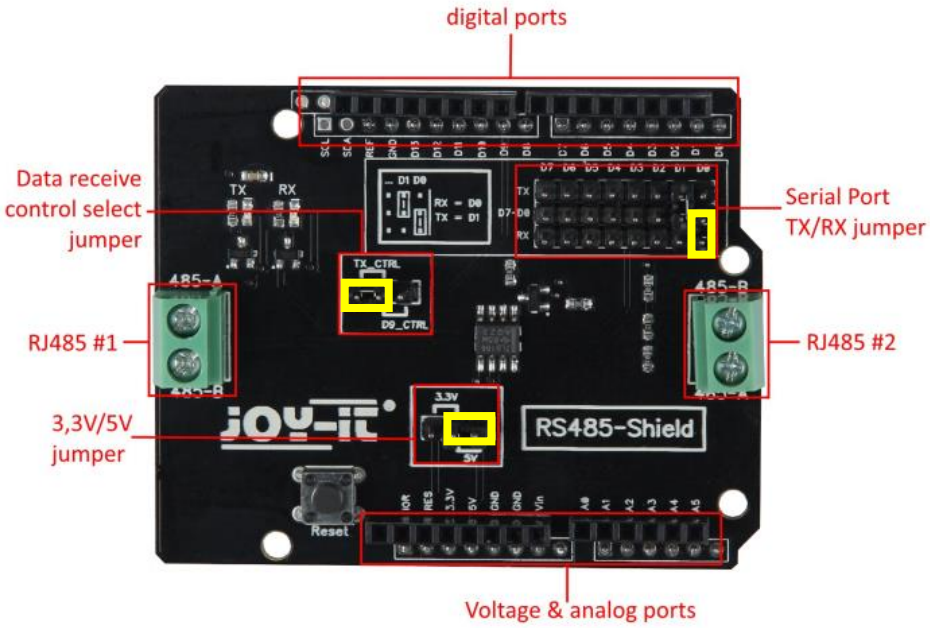


Relais-1: PTT (Eingang Sendetaste am Becker Funkgerät)
Relais-2: PCL (Steuerung Beleuchtung)
Relais-3+4: unbenutzt

RS485-Interface: Joy-It RS485-Shield

https://www.reichelt.de/arduino-shield-rs485-7lb184-pcd-shd-rs485-p151978.html?&trstct=pos_0&nbc=1



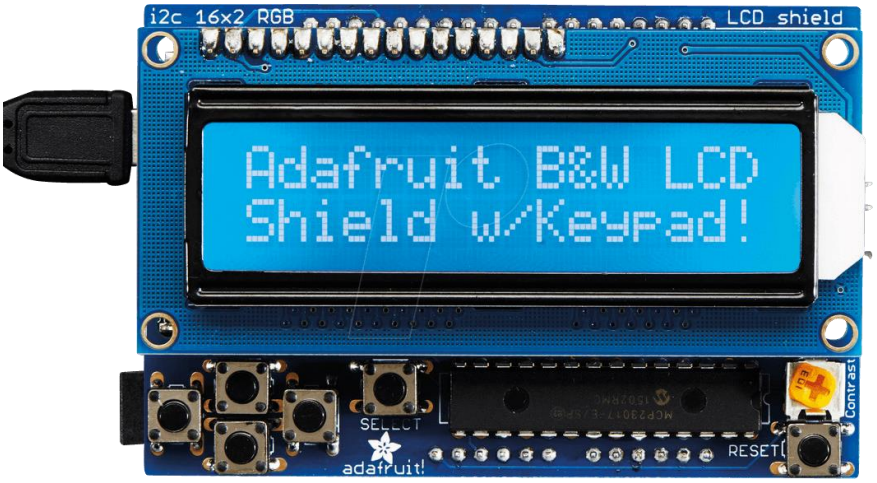


Jumper-Konfiguration:

TX_CTRL:	gesteckt
5V:	gesteckt
D0-RX:	gesteckt (muss zum Upload des Sketches entfernt werden)

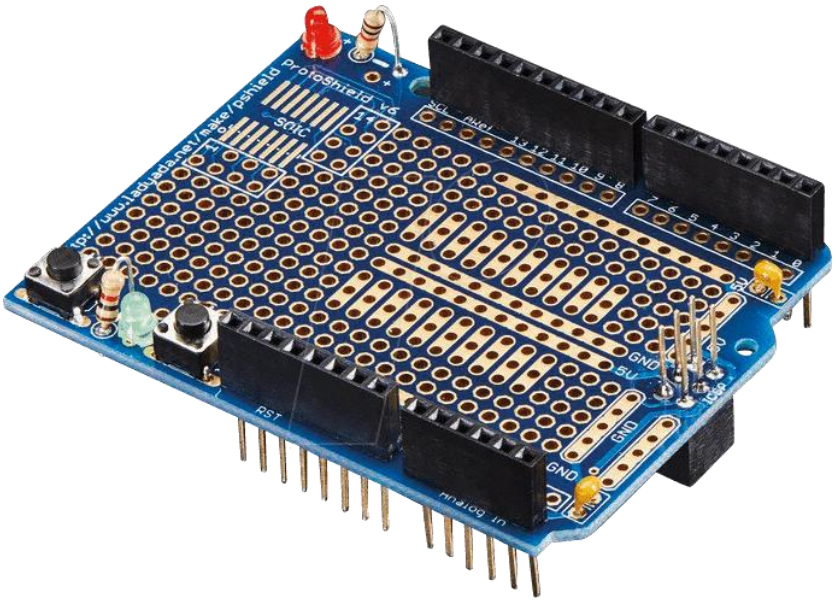
Display: Adafruit LCD Shield Kit with 16x2 Character Display

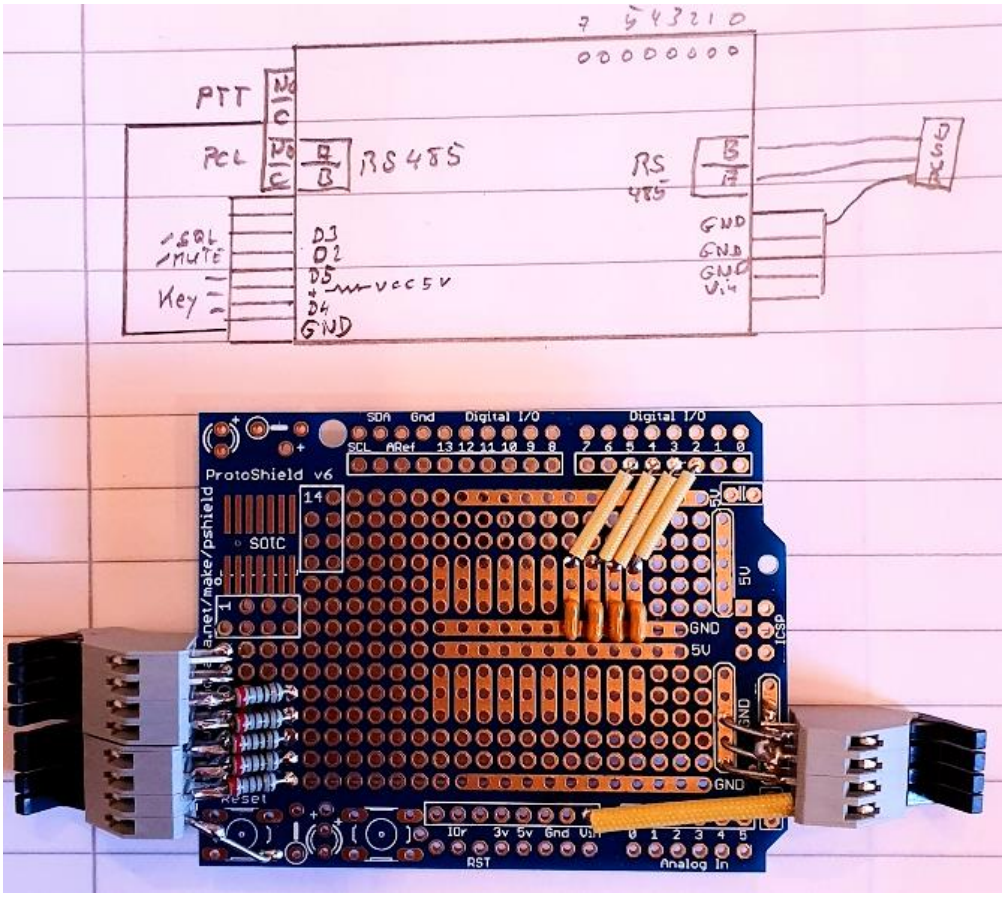
https://www.reichelt.de/arduino-shield-display-lcd-kit-16x2-blau-weiss-arduino-shd-lcd-p159967.html?&nbc=1&trstct=lsbght_sldr::270527



Anschlüsse: Adafruit Prototype Shield No. 2077

https://www.reichelt.de/arduino-shield-protoshield-fuer-arduino-kit-stapelbar-ard-shd-proto-p235460.html?&trstct=pos_7&nbc=1





Audio-Interface zum Becker Funkgerät: Hama Niederfrequenz-Entstörfilter

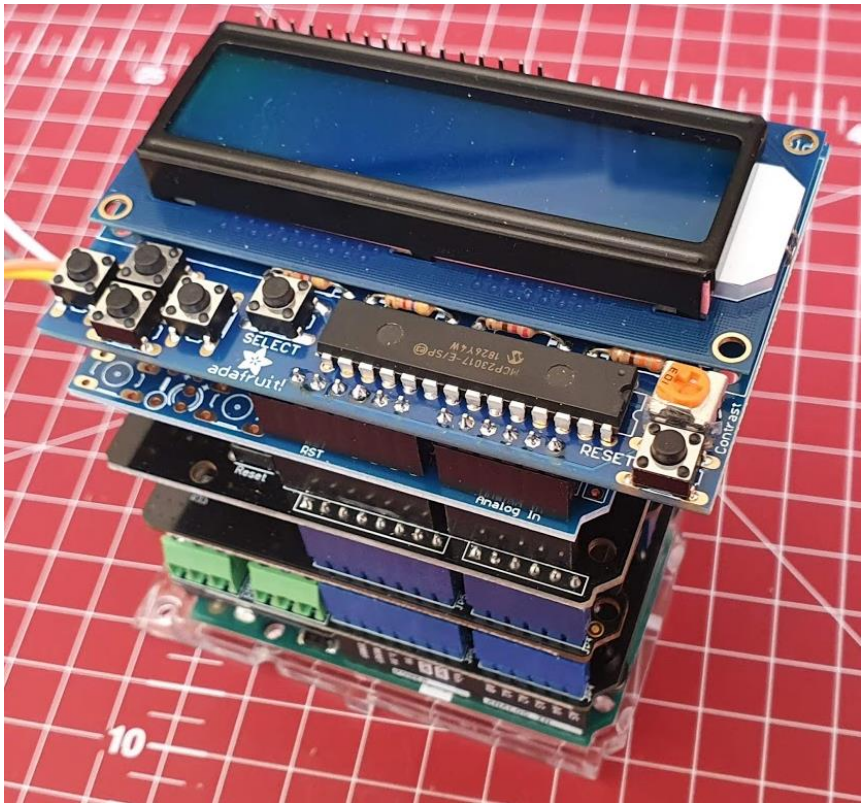
https://de.hama.com/00045683/hama-niederfrequenz-entstoerfilter

MP3-Shield➔männlicher Chinch weiblicher Chinch➔Becker

Aufbau

Steckreihenfolge

Von unten nach oben: Uno – Relay – Audio – RS485 – Prototyp – LCD



Zuordnung der Arduino Ein-/Ausgänge

Eingänge

D0 - RX Data from Weather Sensor
D1 - TX Debug Data to USB Serial Port
D2 - Squelch output from Radio (low active)
D3 - Mute Weather announcement (low active)
D4 - RC/Modellflug Key (high active)
D5 - KFZ Key (high active)

Ausgänge

D6 - Relays 2 - PCL - Pilot Controlled Lighting
D7 - Relays 1 - PTT - Push To Talk input of Radio

Audio/MP3-Shield

Play Control:

- D3 - Receiving signal from button for Volume Up.
- D4 - Receiving signal from switch for Next Song function.
- D5 - Receiving signal from switch for Play&Stop and Record function.
- D6 - Receiving signal from switch for Previous Song function.
- D7 - Receiving signal from button for Volume Down.
- D8 - Green Led instructions.

SPI Interface:

- D10 - SPI Chip Select
- D11 - SPI MOSI
- D12 - SPI MISO
- D13 - SPI SCK

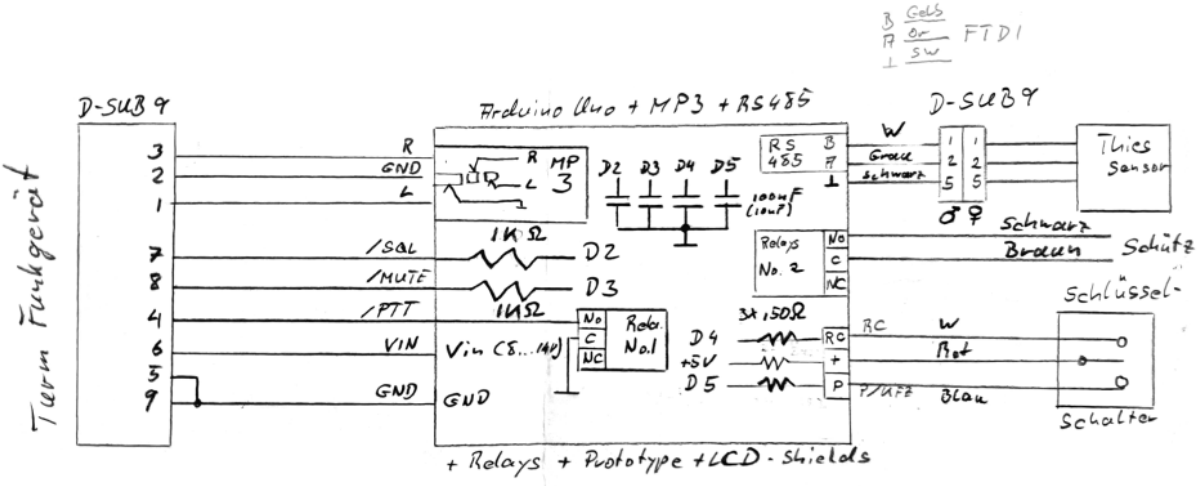
VS1053 Interface:

- A0 - Reset of VS1053
- A1 - Data Require of VS1053
- A2 - Data Select of VS1053
- A3 - Chip Select of VS1053

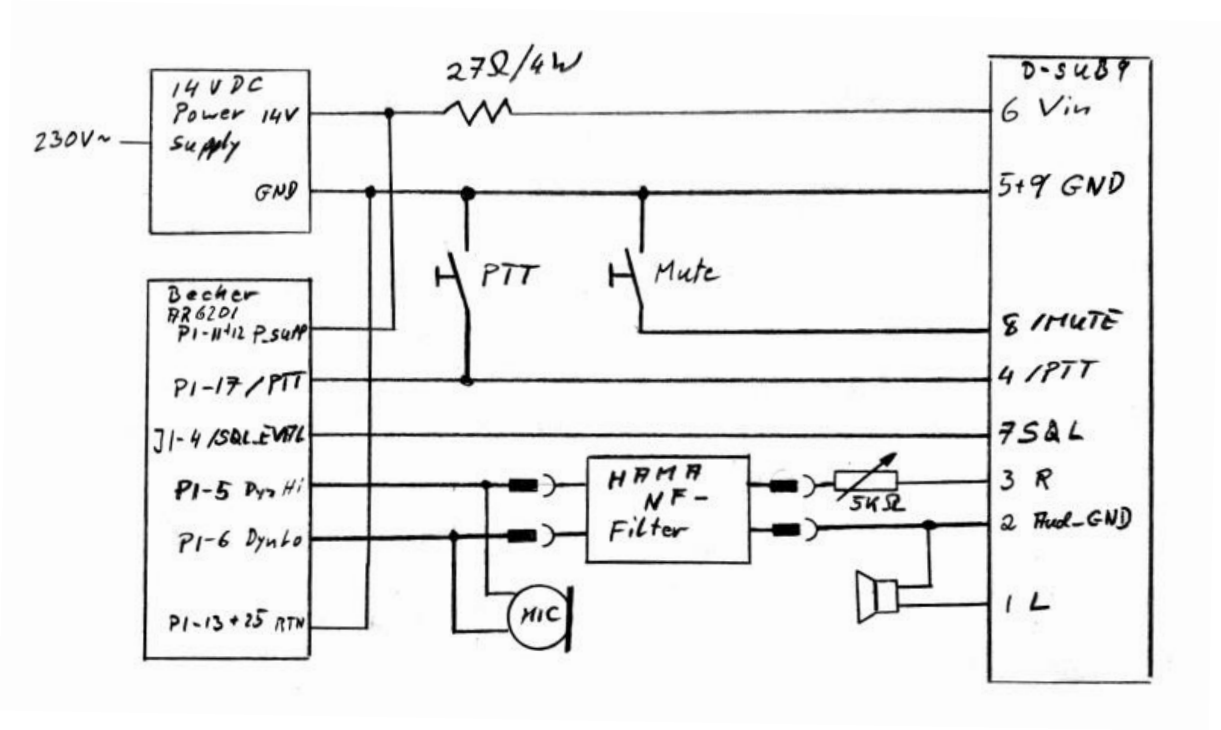
Relays shield

- D4 - Relays 4
- D5 - Relays 3
- D6 - Relays 2
- D7 - Relays 1

Verdrahtung (Arduino)



Verdrahtung (Becker AR6201)



Stromaufnahme

Kein Relais angezogen: 90 mA
Ein Relais angezogen: 150 mA
Zwei Relais angezogen: 215 mA
(an Pin 6 des D-SUB-9)

Software (Arduino Sketch)

Quellcode

```
#include <TimeLib.h>
#include <SD.h>            // for MP3 replay
#include <SPI.h>           // for MP3 replay
#include <arduino.h>       // for MP3 replay
#include <MP3-Shield.h>   // for MP3 replay
#include <avr/wdt.h>      // for Watch-Dog Support

// Adafruit RGB Character LCD Shield and Library
#include <Wire.h>
#include <Adafruit_RGBLCDShield.h>
#include <utility/Adafruit_MCP23017.h>

// These #defines make it easy to set the backlight color
#define BL_OFF 0x0
#define RED 0x1
#define YELLOW 0x3
#define GREEN 0x2
#define TEAL 0x6
#define BLUE 0x4
#define VIOLET 0x5
#define WHITE 0x7
//**********************

// The shield uses the I2C SCL and SDA pins. On classic Arduinos
// this is Analog 4 and 5 so you can't use those for analogRead() anymore
// However, you can connect other I2C sensors to the I2C bus and share
// the I2C bus.
Adafruit_RGBLCDShield lcd = Adafruit_RGBLCDShield();

/* ****************************
inputs:
D0 - RX Data from Weather Sensor
D1 - TX Debug Data to USB Serial Port
D2 - Squelch output from Radio
D3 - Mute Weather announcement
D4 - RC/Modellflug Key
D5 - KFZ Key

outputs:
D6 - Relays 2 - PCL - Pilot Controlled Lighting
D7 - Relays 1 - PTT - Push To Talk input of Radio

inputs used by MP3-Shield:

Pins used for Play Control:
D3 - Receiving signal from button for Volume Up.
D4 - Receiving signal from switch for Next Song function.
D5 - Receiving signal from switch for Play&Stop and Record function.
D6 - Receiving signal from switch for Previous Song function.
D7 - Receiving signal from button for Volume Down.
D8 - Green Led instructions.

Pins Used for SPI Interface:
D10 - SPI Chip Select
D11 - SPI MOSI
D12 - SPI MISO
D13 - SPI SCK

Pins Used for VS1053 Interface:
A0 - Reset of VS1053
A1 - Data Require of VS1053
A2 - Data Select of VS1053
A3 - Chip Select of VS1053

outputs assigned to Relays shield:
D4 - Relays 4
D5 - Relays 3
D6 - Relays 2
D7 - Relays 1

Info (millis-rollover): https://www.faludi.com/2007/12/18/arduino-millis-rollover-handling/

*******************************/

// defines:

#define default_volume 10  // default MP3 volume, 0 = max, 254 = min

#define SQL_PIN  2   // the number of the Squelch pin
#define MUTE_PIN 3   // pin to mute weather announcement
#define RC_PIN   4   // RC / Modelllfug Ops
#define KFZ_PIN  5   // KFZ Test Ops
#define PTT_PIN  7   // PTT Relay
#define PCL_PIN  6   // PCL Relay
#define GRN_LED  8   // GREEN LED (used by MP3-Player)

// Constants:

const int PTT_DELAY = 500;  // 500 ms MP3 delay after PTT release

const int PCL_MIN_PRESS_TIME   =  100;  //  200 milliseconds (for PCL)
const int PCL_SHORT_PRESS_TIME = 1000;  // 1000 milliseconds (for PCL)
const int PCL_LONG_PRESS_TIME  = 7000;  // 7000 milliseconds (for weather announcement)

const int OPS_MIN_PRESS_TIME   =  100;  //  100 milliseconds (for ops announcement key)
const int OPS_SHORT_PRESS_TIME = 2000;  // 2000 milliseconds (for ops announcement key)
const int OPS_LONG_PRESS_TIME  = 3000;  // 3000 milliseconds (for ops announcement key)
```

```
const int MUTE_MIN_PRESS_TIME   =  100; //  100 milliseconds (for muting button)
const int MUTE_SHORT_PRESS_TIME = 1000; // 1000 milliseconds (for muting button)
const int MUTE_LONG_PRESS_TIME  = 2000; // 2000 milliseconds (for muting button)
const int MUTE_RESET_PRESS_TIME = 5000; // 5000 milliseconds (for muting button)

const unsigned long PTTTimeout = 1500;    // timeout for PTT counting 1.5 sec
const unsigned int  PCLTimeout = 1800;    // PCL on time = 30 minutes
const unsigned int  PCLWarnTimeout = 60;  // PCL warining 1 minute befor turning lights off

const unsigned char Mute_Auto_Off_Hrs=18; // time (18:00 hrs UTC) to automatically unmute

const unsigned long RC_KFZ_Time = 14400;  // 4 hours

const unsigned char SensTimeout = 30;     // 60 seconds

const unsigned char PCL_ON_Klicks   = 3;       // 3 clicks to turn lights on
const unsigned char PCL_OFF_Klicks  = 5;       // 5 clicks to turn lights off
const unsigned char MUTE_OFF_Klicks = 7;       // 7 clicks to un-mute weather announcement

const unsigned int WeatherTimeout = 180;  // 3 minutes (weather announcements blocked for this time)

const float WCA = 20;                // wind sensor correction angle = +20°

const int QNH_Offset = 42;               // to compensate elevation of airfield (43 hPa = ~1170 ft)

const char STX = 2;                      //ASCII-Code 02, text representation of the STX code
const char ETX = 3;                      //ASCII-Code 03, text representation of the ETX code

// Variables will change:
unsigned char lastState_port_D    = bit(SQL_PIN) + bit(MUTE_PIN);  // the previous state Port-D
unsigned char currentState_port_D = bit(SQL_PIN) + bit(MUTE_PIN);  // the current reading of Port-D

unsigned long pressedTime  = 0;
unsigned long releasedTime = 0;

#define RC_pressedTime  pressedTime     // use pressed/releasedTime variables too
#define RC_releasedTime releasedTime
#define KFZ_pressedTime  pressedTime
#define KFZ_releasedTime releasedTime

unsigned long WeatherTime = 0;           // keep time of last weather announcement

int PTTCounter = 0;                      // counts number of PPT clicks (e.g. 3 = runway lighting on, 5 = ...)
unsigned long PTTStart = 0;              // Timeout counter for PCL

unsigned long PCLTime = 0;               // keep time of switching on the lights
unsigned char PCL = 0;                   // 1 = lights on, 0 = light 0ff
unsigned char PCL_Warning = 1;           // will be set to 0 after transmitting the "one minute lights off" warning
unsigned char Mute = 0;                  // mute anoutcement

unsigned char RC_KFZ = 0;                // 0 = no special operations, 1 = RC, 2 = KFZ
unsigned long RC_KFZ_Off_Time = 0;       // special ops timeout

float windspeed = 0;                     // actual readings (1 Hz update rate)
unsigned int winddir = 0;                // actual readings (1 Hz update rate)
unsigned int pressure = 0;               // actual readings (1 Hz update rate)

//float temperature = 0, humidity = 0;      // currently not processed
// float Temperature=0;

float WindSpeed=0;                       // filtered (low pass) values
float UX = 0.0, UY = 0.0;                // filtered (low pass) values

unsigned char SensorOKAY = 0;            // 1 = weather sensor data OKAY
unsigned char new_weather = 0;           // 1 = new weather data received
unsigned char SensorTimeout = 0;         // counter for sensor timeout

long  Time_Last_Processing = 0;          // stores last UNIX time of processing weather data

playingstatetype playerState;            // used to check state of MP3 player

// code:

void setup()
{
  Serial.begin(9600);
  Serial.setTimeout(100);   // 100 ms serial timeout (weather frame ~50ms)

  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  lcd.setBacklight(WHITE);
  lcd.setCursor(0, 0);
  lcd.print(F("  Michelstadt  "));
  lcd.setCursor(0, 1);
  lcd.print(F("WetteransageRevE"));

  pinMode(SQL_PIN, INPUT_PULLUP);
  pinMode(MUTE_PIN, INPUT_PULLUP);
  pinMode(RC_PIN, INPUT);
  pinMode(KFZ_PIN, INPUT);
  pinMode(PTT_PIN, OUTPUT);
  digitalWrite(PTT_PIN, LOW);
  pinMode(PCL_PIN, OUTPUT);
  digitalWrite(PCL_PIN, LOW);

  Serial.println();
  Serial.println(F("Reset, Time not set"));
  setTime(1357041600);  // 12:00
  //setTime(1357041500); // vor 11:58

  player.begin();                     //will initialize the hardware and set default mode to be normal.
  player.setVolume(default_volume);
  digitalWrite(PTT_PIN, HIGH);
```

```
  delay(PTT_DELAY);
  player.setPlayMode(PM_NORMAL_PLAY); //
  player.addToPlaylist((char*)("nix.mp3"));
  player.addToPlaylist((char*)("wakt.mp3"));
  WeatherTime = now() - WeatherTimeout;
  PCLTime = now()+PCLTimeout;
  delay(2000);
  wdt_enable(WDTO_8S);              // Watch-Dog activated, 8 second trip time
}

void LCD_RC_KFZ(unsigned char RC_KFZ_State){
  lcd.setCursor(15, 0);
  if (RC_KFZ_State == 0) lcd.print(F(" "));
  if (RC_KFZ_State == 1) lcd.print(F("M"));
  if (RC_KFZ_State == 2) lcd.print(F("P"));
}

void LCD_PCL(unsigned char PCL_State){
  lcd.setCursor(15, 1);
  if (PCL_State)
    lcd.print(F("B"));
  else
    lcd.print(F(" "));
}

void LCD_MUTE(unsigned char Mute_State){
  lcd.setCursor(14, 1);
  if (Mute_State)
      lcd.print(F("X"));
  else
    lcd.print(F(" "));
}

void loop(){
  //wdt_reset();  // reset watch-dog at the begin of every main loop execution
  if (Serial.available()) {
    processSyncMessage();   // e.g. T1357041600
  }

  if ((now() > (Time_Last_Processing + 1 )) ||  new_weather  ){
    Time_Last_Processing = now();               // save time of processing the weather
    new_weather = 0;
    if (SensorTimeout) SensorTimeout--;

    if (timeStatus()!= timeNotSet) {
      Serial.println();
      Serial.print(F("SYST:"));
      Serial.print(now());
      Serial.print(F(" "));
      ProcessWeather(1);      // process wind date, 1 = vector based, 0 = direction based
      Serial.print(F(" B:"));
      Serial.print(PCL);
      Serial.print(F(" BT:"));
      if ((now() - PCLTime) < PCLTimeout ) Serial.print(PCLTimeout - now() + PCLTime);
      else Serial.print(0);
      Serial.print(F(" M:"));
      Serial.print(Mute);
      Serial.print(F(" WT:"));
      if ((now() - WeatherTime) < WeatherTimeout ) Serial.print(WeatherTimeout - now() + WeatherTime);
      else Serial.print(0);
      if (RC_KFZ){
        Serial.print(F(" OPS:"));
        Serial.print(RC_KFZ);
        Serial.print(F(" "));
        Serial.print(RC_KFZ_Off_Time - now());
      }
      Serial.print(F(" MP3:"));
      Serial.print(playerState);
      Serial.print(F(" ST:"));
      Serial.print(SensorTimeout);
      if (Mute)
        if (hour() >= Mute_Auto_Off_Hrs) {
          Mute = 0;
          LCD_MUTE(Mute);
          Serial.print(F(" Mute Off"));
        }
    }
  }
  //delay(10);
  player.play(playerState);  // process MP3-Player

  // process inputs
  // read Port-D
  currentState_port_D = PIND;

  // process SQL Input (active low):
  if((lastState_port_D & bit(SQL_PIN)) && !(currentState_port_D & bit(SQL_PIN)))        // button is pressed
    pressedTime = millis();
  else if(!(lastState_port_D & bit(SQL_PIN)) && (currentState_port_D & bit(SQL_PIN))) { // button is released
    releasedTime = millis();

    int pressDuration = (int)(releasedTime - pressedTime);
    Serial.print(F(" press_duration:"));
    Serial.print(pressDuration);

     if( (pressDuration < PCL_SHORT_PRESS_TIME) &&(pressDuration > PCL_MIN_PRESS_TIME)  ){
      Serial.print(F(" SKP"));
      lcd.setCursor(14, 0);
      lcd.print(F("S"));
      PTTCounter++;
      if (PTTCounter > 0) {
        PTTStart = releasedTime;    // start timeout counter and increase PTT Counter
      }
    }
```

```
  if( pressDuration > PCL_LONG_PRESS_TIME ){
    Serial.print(F(" LKP"));
    lcd.setCursor(14, 0);
    lcd.print(F("L"));
    if (Mute == 0)
      if ((now() - WeatherTime ) > WeatherTimeout ){
        WeatherTime = now();
        lcd.setCursor(14, 0);
        lcd.print(F("W"));
        Say_Weather();
      }
  }
}

// process Mute Input (active low):
if((lastState_port_D & bit(MUTE_PIN)) && !(currentState_port_D & bit(MUTE_PIN)))        // button is pressed
  pressedTime = millis();
else if(!(lastState_port_D & bit(MUTE_PIN)) && (currentState_port_D & bit(MUTE_PIN))) { // button is released
  releasedTime = millis();

  int pressDuration = (int)(releasedTime - pressedTime);
  if( (pressDuration < MUTE_SHORT_PRESS_TIME) && (pressDuration > MUTE_MIN_PRESS_TIME) ){
    Serial.print(F(" Mute On"));
    lcd.setCursor(14, 0);
    lcd.print(F("S"));
    Mute = 1;
    LCD_MUTE(Mute);
  }
  if(( pressDuration > MUTE_LONG_PRESS_TIME ) && ( pressDuration < MUTE_RESET_PRESS_TIME )){
    Serial.print(F(" Mute Off"));
    lcd.setCursor(14, 0);
    lcd.print(F("L"));
    Mute = 0;
    LCD_MUTE(Mute);
  }
  if( pressDuration > MUTE_RESET_PRESS_TIME ){
    Serial.print(F(" APIS Reset, Mute Off, Lights Off, Special Ops Off"));
    lcd.setCursor(14, 0);
    lcd.print(F("R"));
    Mute = 0;
    LCD_MUTE(Mute);
    PCL = 0;
    PCLTime = 0;
    digitalWrite(PCL_PIN, LOW);
    PCL_Warning = 1;
    LCD_PCL(PCL);
    Say_PCL_Off();
    RC_KFZ = 0;               // 0 = no special operations, 1 = RC, 2 = KFZ
    LCD_RC_KFZ(RC_KFZ);
    RC_KFZ_Off_Time = 0;
    WeatherTime = now() - WeatherTimeout;  // reset weather time
    //Say_OPS();
  }
}

// process RC - Modellflug key (active high) *********************
if(!(lastState_port_D & bit(RC_PIN)) && (currentState_port_D & bit(RC_PIN)))        // button is pressed
  RC_pressedTime = millis();
else if((lastState_port_D & bit(RC_PIN)) && !(currentState_port_D & bit(RC_PIN))) { // button is released
  RC_releasedTime = millis();
  int pressDuration = (int)(RC_releasedTime - RC_pressedTime);
  if( pressDuration > OPS_LONG_PRESS_TIME ){
    Serial.print(F(" RC-Key-Long"));
    RC_KFZ = 0;               // 0 = no special operations, 1 = RC, 2 = KFZ
    LCD_RC_KFZ(RC_KFZ);
    RC_KFZ_Off_Time = 0;
    Say_OPS();
  }
  else
    if( (pressDuration < OPS_SHORT_PRESS_TIME) && (pressDuration > OPS_MIN_PRESS_TIME) ){
      Serial.print(F(" RC-Key-Short"));
      RC_KFZ = 1;               // 0 = no special operations, 1 = RC, 2 = KFZ
      LCD_RC_KFZ(RC_KFZ);
      RC_KFZ_Off_Time = now() + RC_KFZ_Time;
      Say_OPS();
    }
}

// process KFZ Key  (active high) ***************************
if(!(lastState_port_D & bit(KFZ_PIN)) && (currentState_port_D & bit(KFZ_PIN)))        // button is pressed
  KFZ_pressedTime = millis();
else if((lastState_port_D & bit(KFZ_PIN)) && !(currentState_port_D & bit(KFZ_PIN))) { // button is released
  KFZ_releasedTime = millis();
  int pressDuration = (int)(KFZ_releasedTime - KFZ_pressedTime);
  if( pressDuration > OPS_LONG_PRESS_TIME ){
    Serial.print(F(" KFZ-Key-Long"));
    RC_KFZ = 0;               // 0 = no special operations, 1 = RC, 2 = KFZ
    LCD_RC_KFZ(RC_KFZ);
    RC_KFZ_Off_Time = 0;
    Say_OPS();
  }
  else
    if( (pressDuration < OPS_SHORT_PRESS_TIME) && (pressDuration > OPS_MIN_PRESS_TIME) ){
      Serial.print(F(" KFZ-Key-Short"));
      RC_KFZ = 2;               // 0 = no special operations, 1 = RC, 2 = KFZ
      LCD_RC_KFZ(RC_KFZ);
      RC_KFZ_Off_Time = now() + RC_KFZ_Time;
      Say_OPS();
    }
}

// save the last port-D state
lastState_port_D = currentState_port_D;
```

```
  if ((now()> RC_KFZ_Off_Time) && RC_KFZ ){
    Serial.print(F(" KFZ-RC Off"));
    RC_KFZ = 0;                // 0 = no special operations, 1 = RC, 2 = KFZ
    LCD_RC_KFZ(RC_KFZ);
    RC_KFZ_Off_Time = 0;
  }

  if ( ((now() - PCLTime) > (PCLTimeout - PCLWarnTimeout) ) && PCL && PCL_Warning){
        Serial.print(F(" Lights Off in 1 Minute"));
        PCL_Warning = 0;
        Say_PCL_Off_in_1();
  }

  if ( ((now() - PCLTime) > PCLTimeout ) && PCL ){
        PCL = 0;
        digitalWrite(PCL_PIN, LOW);
        PCL_Warning = 1;
        Serial.print(F(" Lights Off"));
        LCD_PCL(PCL);
        Say_PCL_Off();
  }

  if (((millis() - PTTStart) > PTTTimeout) && (PTTCounter>0)){
    if (PTTCounter == PCL_ON_Klicks) {
      PCLTime = now ();
      PCL = 1;
      PCL_Warning = 1;
      digitalWrite(PCL_PIN, HIGH);
      Serial.print(F(" Lights On"));
      LCD_PCL(PCL);
      Say_PCL_On_30();
    }
    if (PTTCounter == PCL_OFF_Klicks) {
        WeatherTime = now() - WeatherTimeout;  // reset weather time
        PCLTime = 0;
        PCL = 0;
        digitalWrite(PCL_PIN, LOW);
        PCL_Warning = 1;
        Serial.print(F(" Lights Off"));
        LCD_PCL(PCL);
        Say_PCL_Off();
    }
    if (PTTCounter == MUTE_OFF_Klicks) {
        WeatherTime = now() - WeatherTimeout;  // reset weather time
        Serial.print(F(" Mute Off"));
        lcd.setCursor(14, 0);
        lcd.print(F("L"));
        Mute = 0;
        LCD_MUTE(Mute);
        Say_Weather();
    }
    PTTCounter = 0;
    PTTStart = 0;
    Serial.print(F(" CLR PPT_CNT"));
  }
  wdt_reset();  // reset watch-dog at the end of every main loop execution
}

void serialFlush(){
  while(Serial.available() > 0) {
    char t = Serial.read();
  }
}

void processSyncMessage() {
  char str[50];
  if(Serial.find(STX)) {
    int tage, monate, jahre, stunden, minuten, sekunden;
    long checksumme = 1;
    int XOR_CS = 0;
    char* ptr = str;              // set ptr to start of string
    String serial_str = Serial.readString();
    serial_str.toCharArray(str, 50);
    //Serial.println(str);

    if (ptr != NULL)
    {
      char *ptr2;               // parse checksum (8-bit hex number in string)
      checksumme = strtol(ptr+46, &ptr2, 16);
      for (int i = 0; i < 45; i++) {    // calculate XOR checksum
        XOR_CS ^= str[i];
      }
    }
    //strcat(str, "001.4 272 -03.3  94  985.0 23.03.21 06:59:46 *27"); // append wind sensor string
    // test string: !001.4 272 +03.3  94  985.0 23.03.21 06:59:46 *27
    // test string: !001.5 285 +03.3  94  985.0 23.03.21 06:59:28 *26
    // test string: !001.4 272 -03.3  94  985.0 23.03.21 06:59:46 *21
    //              !011.5 085 +03.3  94  985.0 23.03.21 06:59:28 *25
    //              !001.5 085 +03.3  94  985.0 23.03.21 06:59:28 *24

    if (XOR_CS == checksumme){
      if (ptr != NULL)
        windspeed = atof(ptr) * 2;       // m/s * 2 = kts
      if (ptr != NULL)
        winddir = atoi(ptr + 6);
      //if (ptr != NULL)
      //  temperature = atof(ptr + 10);
      //if (ptr != NULL)
      //  humidity = atof(ptr + 16);
      if (ptr != NULL)
        pressure = QNH_Offset + atoi(ptr + 20);
      if (ptr != NULL)
        tage = atoi(ptr + 27);
      if (ptr != NULL)
```

```
        monate = atoi(ptr + 30);
      if (ptr != NULL)
        jahre = atoi(ptr + 33);
      if (ptr != NULL)
        stunden = atoi(ptr + 36);
      if (ptr != NULL)
        minuten = atoi(ptr + 39);
      if (ptr != NULL)
        sekunden = atoi(ptr + 42);

      setTime(stunden, minuten, sekunden, tage, monate, jahre);  //int hr,int min,int sec,int day, int month, int yr
      //Serial.print(F("CS OKAY ==> Time Set"));

      if ((stunden == 0) && (minuten == 0) && (sekunden >= 52)) while(1){} // Reset Arduino via Watch-Dog at UTC==00:00:52
(once a day)

      new_weather = 1;
      SensorTimeout = SensTimeout;

      if (SensorOKAY == 0){
        Say_Sensor_OKAY ();
        SensorOKAY = 1;
      }
    }
    else {
      Serial.println();
      Serial.print(F("CS Error ==> Time + Weather not set"));
    }
  }
  else Serial.print(F("STX Error"));
  serialFlush();
}

float lowPass(float  input, float output ) {
    output = output + 0.05 * (input - output);
      return output;
  }

void LCD_4_Places(unsigned int number){
  if (number < 9999) {
    if (number < 1000) lcd.print(F(" "));
    if (number < 100) lcd.print(F("0"));
    if (number < 10) lcd.print(F("0"));
    lcd.print(number, DEC);
  }
}

void LCD_3_Places(unsigned int number){
  if (number < 999) {
    if (number < 100) lcd.print(F("0"));
    if (number < 10)  lcd.print(F("0"));
    lcd.print(number, DEC);
  }
}

void LCD_2_Places(unsigned int number){
  if (number < 99) {
    if (number < 10) lcd.print(F("0"));
    lcd.print(number, DEC);
  }
}

void LCDClockDisplay(){
  // digital clock display of the time hh:mm
  LCD_2_Places((unsigned int)hour());
  lcd.print(F(":"));
  LCD_2_Places((unsigned int)minute());
  //lcd.print(F(" "));
}

void ProcessWeather(bool vector_based) {

  if (vector_based) {
    UX = lowPass( windspeed * cos(((float)winddir+WCA) * DEG_TO_RAD), UX);     // wind vector
    UY = lowPass( windspeed * sin(((float)winddir+WCA) * DEG_TO_RAD), UY);     // wind vector
  }
  else {
    UX = lowPass( cos(((float)winddir+WCA) * DEG_TO_RAD), UX);                   // wind direction only
    UY = lowPass( sin(((float)winddir+WCA) * DEG_TO_RAD), UY);                   // wind direction only
  }
  if (SensorTimeout){
    //WindDir = (int)((RAD_TO_DEG * atan2(UY, UX))+360) % 360;       // filtering wind direction / vector
    WindSpeed = lowPass(windspeed, WindSpeed);                      // filtering wind speed
  }
  else {
    UX = 1.0f;
    UY = 0.0f;
    WindSpeed = 0;
    pressure = 0;
  }

  Serial.print(F("V:")); Serial.print(WindSpeed, 1);
  Serial.print(F(" W:")); Serial.print((int)((RAD_TO_DEG * atan2(UY, UX))+360) % 360);
  Serial.print(F(" QNH:")); Serial.print(pressure, DEC);

  lcd.setCursor(0, 0);
  lcd.print(F("Wind:"));
  lcd.setCursor(5, 0);
  LCD_3_Places((unsigned int)((RAD_TO_DEG * atan2(UY, UX))+360) % 360);
  lcd.print((char)0xDF);                // 0xDF = °
  //lcd.print(F(" "));
  lcd.setCursor(9, 0);
  LCD_2_Places((unsigned int)WindSpeed);
  lcd.print(F("kts "));
```

```
  lcd.setCursor(0, 1);
  lcd.print(F("QNH:"));
  LCD_4_Places(pressure);
  lcd.print(F(" "));
  LCDClockDisplay();
  LCD_RC_KFZ(RC_KFZ);
  LCD_PCL(PCL);
  LCD_MUTE(Mute);
}

void Say_Digit(int digit){
  switch (digit) {
  case 0:
    player.addToPlaylist((char*)("0.mp3"));
    break;
  case 1:
    player.addToPlaylist((char*)("1.mp3"));
    break;
  case 2:
    player.addToPlaylist((char*)("2.mp3"));
    break;
  case 3:
    player.addToPlaylist((char*)("3.mp3"));
    break;
  case 4:
    player.addToPlaylist((char*)("4.mp3"));
    break;
  case 5:
    player.addToPlaylist((char*)("5.mp3"));
    break;
  case 6:
    player.addToPlaylist((char*)("6.mp3"));
    break;
  case 7:
    player.addToPlaylist((char*)("7.mp3"));
    break;
  case 8:
    player.addToPlaylist((char*)("8.mp3"));
    break;
  case 9:
    player.addToPlaylist((char*)("9.mp3"));
    break;
  default:
    // Statement(s)
    break; // Wird nicht benötigt, wenn Statement(s) vorhanden sind
  }
}

void Say_Simple_Wind(float Vin, int Win){
  float WW;
  int W;
  WW = Win + 0.5f;
  WW = (WW + 22.5f) / 45.0f;
  W = (int)WW;

  if ((Vin >= 2.0f) && (Vin < 5.0f)) player.addToPlaylist((char*)("schw.mp3"));
  if ((Vin >= 5.0f) && (Vin < 10.0f)) player.addToPlaylist((char*)("msgw.mp3"));
  if (Vin >= 10.0f) player.addToPlaylist((char*)("stkw.mp3"));
  if (Vin < 2.0f) player.addToPlaylist((char*)("still.mp3"));
  else
  {
     switch (W) {
    case 8:
      player.addToPlaylist((char*)("n.mp3"));
      break;
    case 0:
      player.addToPlaylist((char*)("n.mp3"));
      break;
    case 1:
      player.addToPlaylist((char*)("no.mp3"));
      break;
    case 2:
      player.addToPlaylist((char*)("o.mp3"));
      break;
    case 3:
      player.addToPlaylist((char*)("so.mp3"));
      break;
    case 4:
      player.addToPlaylist((char*)("s.mp3"));
      break;
    case 5:
      player.addToPlaylist((char*)("sw.mp3"));
      break;
    case 6:
      player.addToPlaylist((char*)("w.mp3"));
      break;
    case 7:
      player.addToPlaylist((char*)("nw.mp3"));
      break;
      default:
      // Statement(s)
      break; // Wird nicht benötigt, wenn Statement(s) vorhanden sind
    }
  }
}

void Say_Mixed_Wind(float Vin, int Win){
  float WW;
  int W, V;
  WW = Win + 0.5f;
  WW = (WW + 22.5f) / 45.0f;
  W = (int)WW;
```

```
  V = Vin;
  player.addToPlaylist((char*)("wndm.mp3"));

  switch (V) {
    case 0:
      player.addToPlaylist((char*)("0.mp3"));
      break;
    case 1 ... 2:
      player.addToPlaylist((char*)("2.mp3"));
      break;
    case 3 ... 4:
      player.addToPlaylist((char*)("4.mp3"));
      break;
    case 5 ... 6:
      player.addToPlaylist((char*)("6.mp3"));
      break;
    case 7 ... 8:
      player.addToPlaylist((char*)("8.mp3"));
      break;
    case 9 ... 10:
      player.addToPlaylist((char*)("10.mp3"));
      break;
    case 11 ... 12:
      player.addToPlaylist((char*)("12.mp3"));
      break;
    case 13 ... 14:
      player.addToPlaylist((char*)("14.mp3"));
      break;
    case 15 ... 16:
      player.addToPlaylist((char*)("16.mp3"));
      break;
    case 17 ... 18:
      player.addToPlaylist((char*)("18.mp3"));
      break;
    case 19 ... 22:
      player.addToPlaylist((char*)("20.mp3"));
      break;
    case 23 ... 27:
      player.addToPlaylist((char*)("25.mp3"));
      break;
    case 28 ... 32:
      player.addToPlaylist((char*)("30.mp3"));
      break;
    case 33 ... 37:
      player.addToPlaylist((char*)("35.mp3"));
      break;
    case 38 ... 42:
      player.addToPlaylist((char*)("40.mp3"));
      break;
    case 43 ... 47:
      player.addToPlaylist((char*)("45.mp3"));
      break;
    case 48 ... 1000:
      player.addToPlaylist((char*)("50.mp3"));
      break;
    default:
      // Statement(s)
      break; // Wird nicht benötigt, wenn Statement(s) vorhanden sind
  }

  player.addToPlaylist((char*)("kaus.mp3"));

  switch (W) {
    case 8:
      player.addToPlaylist((char*)("n.mp3"));
      break;
    case 0:
      player.addToPlaylist((char*)("n.mp3"));
      break;
    case 1:
      player.addToPlaylist((char*)("no.mp3"));
      break;
    case 2:
      player.addToPlaylist((char*)("o.mp3"));
      break;
    case 3:
      player.addToPlaylist((char*)("so.mp3"));
      break;
    case 4:
      player.addToPlaylist((char*)("s.mp3"));
      break;
    case 5:
      player.addToPlaylist((char*)("sw.mp3"));
      break;
    case 6:
      player.addToPlaylist((char*)("w.mp3"));
      break;
    case 7:
      player.addToPlaylist((char*)("nw.mp3"));
      break;
    default:
      // Statement(s)
      break; // Wird nicht benötigt, wenn Statement(s) vorhanden sind
  }
}

void Say_WindDir(int number){
  int rounded_number = 0;
  rounded_number = (number + 5) / 10 * 10;
  if (rounded_number >= 360) rounded_number = 0;
  for (int i=1000; i >= 1; i=i/10){
    Say_Digit(rounded_number % i / (i/10));
  }
}
```

```
void Say_WindSpeed(int number){
  int i;
  if (number < 100)
  {
    i = number % 100 /10;
    if (i > 0) {
      Say_Digit(i);
      i = number - i * 10;
      Say_Digit(i);
    }
    else {
      i = number - i * 10;
      if (i == 1) player.addToPlaylist((char*)("nem.mp3"));
      else Say_Digit(i);
    }
  }
}

void Say_QNH(unsigned int number){
  int rounded_number = 0;
  rounded_number = number;
  for (int i=10000; i >= 1; i=i/10){
    if (!((i == 10000)&&((rounded_number % i / (i/10)==0))))
    Say_Digit(rounded_number % i / (i/10));
  }
}

void Prepare_MP3 (){
  do {
    delay(10);
    player.play(playerState);  // process MP3-Player
  } while (playerState != PS_IDLE);
  player.initializePlaylist();
  player.setVolume(default_volume);
  player.setPlayMode(PM_NORMAL_PLAY);
  digitalWrite(PTT_PIN, HIGH);
  delay(PTT_DELAY);
}

void Say_Weather (){
  Serial.print(F(" Say Weather"));
  Prepare_MP3();
  player.addToPlaylist((char*)("radi.mp3"));
  if (SensorTimeout){
    //Say_Simple_Wind(WindSpeed, WindDir);
    Say_Mixed_Wind(WindSpeed, (int)((RAD_TO_DEG * atan2(UY, UX))+360) % 360 );
    player.addToPlaylist((char*)("qnh.mp3"));
    Say_QNH(pressure);
  }
  else {
    player.addToPlaylist((char*)("serr.mp3"));
  }
  player.addToPlaylist((char*)("ende.mp3"));
  //player_void(); // player.addToPlaylist((char*)("paus.mp3"));
  if (RC_KFZ==1) player.addToPlaylist((char*)("rc.mp3"));
  if (RC_KFZ==2) player.addToPlaylist((char*)("kfz.mp3"));
}


void Say_OPS (){
  Prepare_MP3();
  player.addToPlaylist((char*)("radi.mp3"));
  if (RC_KFZ==1) player.addToPlaylist((char*)("rc.mp3"));
  if (RC_KFZ==2) player.addToPlaylist((char*)("kfz.mp3"));
}

void Say_Sensor_OKAY (){
  lcd.setCursor(0, 0);
  lcd.print(F("  Wetterdaten   "));
  lcd.setCursor(0, 1);
  lcd.print(F("   empfangen    "));
  Prepare_MP3();
  player.addToPlaylist((char*)("wdok.mp3"));
}

void Say_PCL_On_30 (){
  Prepare_MP3();
  player.addToPlaylist((char*)("pb30.mp3"));
}

void Say_PCL_Off (){
  Prepare_MP3();
  player.addToPlaylist((char*)("pbas.mp3"));
}

void Say_PCL_Off_in_1 (){
  Prepare_MP3();
  player.addToPlaylist((char*)("pba1.mp3"));
}
```

Arduino Bibliotheken

Die folgenden Arduino Bibliotheken werden benötigt:

arduino.h	Main include file for the Arduino SDK
TimeLib.h	für Zeitmessung / Timeout
SD.h	für Audio/MP3 Wiedergabe
SPI.h	für Audio/MP3 Wiedergabe
MP3-Shield.h	Audio/MP3 Wiedergabe
Wire.h	I²C-Bus
Adafruit_RGBLCDShield.h	LC-Display
Adafruit_MCP23017.h	Port-Expander-IC mit I²C-Bus

MP3-Shield Modifikation

Die Bibliothek „MP3-Shield“ basiert auf der „Music Shield Bibliothek“. Diese benötigt zu viel RAM-Speicher und ist daher wie folgt zu modifizieren:

1. Ordner „MP3-Shield“ unter „user\Documents\Arduino\libraries\..“ anlegen
2. Inhalt der Music Shield Bibliothek in den neuen Ordner kopieren
3. MusicPlayer.cpp und MusicPlayer.h in MP3-shield.cpp bzw. .h umbenennen
4. Änderungen in MusicPlayer.h (Änderungen sind unterstrichen bzw. ~~durchgestrichen~~)

   Zeile 50:
   ```
   #define MAX_SONG_TOTAL_NUM 16
   #define FILE_NAME_LENGTH      3
   ```

   Hinweis: durch die Änderung, werden nur noch die ersten vier Zeichen des MP3-Dateinamens ausgewertet

   Zeile 98:
   ```
   typedef struct songDesc {
       char name[FILE_NAME_LENGTH];                          //was 13
       uint16_t index;
   } song_t;
   typedef struct songsPlaylist {
       song_t p_songFile[MAX_SONG_TOTAL_NUM];
       unsigned char songTotalNum;   //total number of songs in the playlist
       unsigned char currentSongNum;
   } spl_t;
   ```

   Ab Zeile 176: alle midi Prototypen löschen
   ```
   ~~void    midiDemoPlayer(void);         //oliver wang~~
   ~~//for Midi Player~~
   ~~void midiWriteData(byte cmd, byte high, byte low);~~
   ~~void midiNoteOn(byte channel, byte note, byte rate);~~
   ~~void midiNoteOff(byte channel, byte note, byte rate);~~
   ~~void midiSendByte(byte data);~~
   ```

5. Änderungen in MusicPlayer.cpp (Änderungen sind unterstrichen bzw. ~~durchgestrichen~~)

   Zeile 33: ~~#include <MusicPlayer.h>~~ ➔ #include <MP3-Shield.h>

   Zeile 74: auskommentieren
   ```
   //void showString(PGM_P s) {
   //    char c;
   //    while ((c = pgm_read_byte(s++)) != 0) {
   ```

```
//       Serial.print(c);
//    }
//}
```

Zeile 143: auskommentieren

```
//    _keys[0].setPara(KEY_PS, 1, 100, CS_PLAYPAUSE);
//    _keys[1].setPara(KEY_VD, 2, 100, CS_DOWN);
//    _keys[2].setPara(KEY_VU, 2, 100, CS_UP);
//    _keys[3].setPara(KEY_BK, 3, 100, CS_PREV);
//    _keys[4].setPara(KEY_NT, 3, 100, CS_NEXT);
```

ab Zeile 153: alle Zeile mit „`showString(PSTR…`“ und „`Serial.Print…`“ auskommentieren

ab Zeile 199:

```
void MusicPlayer::play(playingstatetype& PlayerState) {
    if (Analog_Enable) {
        scanAnalogSensor();
    }
    if (Digital_Enable) {
        scanDigitalSensor();
    }
   PlayerState = playingState;
   _play();
}
```

Zeile 211:

```
//    song_t* songFile;
    song_t songFile;
```

ab Zeile 356: auskommentieren

```
        // case PS_PRE_RECORD:
            // _preRecording();
            // //showString(PSTR("Recording...\r\n"));
            // recording_state = 0;
            // playingState = PS_RECORDING;
            // break;
        //
        // case PS_RECORDING:
            // //recording
            // _recording();
            // break;
        // //
        // case PS_POST_RECORD:
            // cur_file.close();
            // vs1053.softReset();
            // //showString(PSTR("stop\r\n"));
            // playingState = PS_IDLE;
            // break;
```

ab Zeile 382: folgenden Code auskommentieren oder löschen

```
    /*  SdFile f;
        if (!f.open(&root, songFile, O_READ))
        {
        Serial.print(songFile);
        showString(PSTR(" does not exists.\r\n"));
        return;
        }
        f.close();
        uint16_t index = root.dirIndex();
        if (!_inPlayList(index))
        {
        addToPlaylist(songFile);
        }*/
```

ab Zeile 382: wie folgt ändern

```
    if (addToPlaylist(songFile)) {
        //added by shao
        //switch to this song
        for (int i = 0; i < spl.songTotalNum; i++) {
//            if (strcmp(songFile, spl.p_songFile[i]->name) == 0) {
              if (strcmp(songFile, spl.p_songFile[i].name) == 0) {
                //showString(PSTR("Seeked to "));
                //Serial.println(songFile);
```

```
                spl.currentSongNum = i;
                playingState = PS_PRE_PLAY;
            }
        }
```

ab Zeile 400: `void MusicPlayer::scanAndPlayAll(void)` löschen

ab Zeile 403: `boolean MusicPlayer::_addToPlaylist...` wie folgt ändern

```
boolean MusicPlayer::_addToPlaylist(uint16_t index, char* songName) { //add a song to
current playlist
    if (spl.songTotalNum >= (MAX_SONG_TOTAL_NUM - 1)) {
        return false;
    }

//    if (_inPlayList(index)) {
//        //Serial.print(songName);
//        //showString(PSTR(" already exists in playlist.\r\n"));
//        return true;
//    }

    SdFile f;
    if (!f.open(&root, index, O_READ)) {
        //Serial.print(songName);
        ////showString(PSTR(" cant be opened.\r\n"));
        return false;
    }
    f.close();
    if (1) {
        strncpy(spl.p_songFile[spl.songTotalNum].name, songName, FILE_NAME_LENGTH);
        spl.p_songFile[spl.songTotalNum].index = index;
               //sd->index = index;
        //spl.p_songFile[spl.songTotalNum] = sd;
               //free(sd);
        spl.songTotalNum++;
               //Serial.print("spl.songTotalNum="); //Serial.println(spl.songTotalNum);
        return true;
    } else {
        //showString(PSTR("run out of ram.\r\n"));
    }
    return false;
}
```

MP3-Dateien

Sample-Rate: 44100 Hz
Kanäle: mono
Bitrate: variabel, ca. 100 … 120 kBit/s
MP3-Dateien:

Dateiname	Gesprochener Text
0.mp3	0
1.mp3	1
10.mp3	1 – 0
12.mp3	1 – 2
14.mp3	1 – 4
16.mp3	1 – 6
18.mp3	1 – 8
2.mp3	2
20.mp3	2 – 0
25.mp3	2 – 5
3.mp3	3
30.mp3	3 – 0
35.mp3	3 – 5
4.mp3	4
40.mp3	4 – 0
45.mp3	4 – 5
5.mp3	5
50.mp3	5 – 0
6.mp3	6
7.mp3	7
8.mp3	8
9.mp3	9
aus.mp3	aus
ein.mp3	ein
ende.mp3	vor der Landung ist der Wind und die Hindernisfreiheit der Piste durch Überfliegen des Platzes zu überprüfen
gaus.mp3	Grad aus
grad.mp3	Grad
~~info.mp3~~	~~Michelstadt Info~~
radi.mp3	Michelstadt Radio
kaus.mp3	Knoten aus
kfz.mp3	Achtung: auf dem Fluggelände finden Kraftfahrzeugtestfahrten statt
kts.mp3	Knoten
mit.mp3	mit
msgw.mp3	mäßiger Wind aus
n.mp3	nördlichen Richtungen
nem.mp3	einem
nix.mp3	- (0,7 Sekunden)
no.mp3	nordöstlichen Richtungen
nw.mp3	nordwestlichen Richtungen
o.mp3	östlichen Richtungen
paus.mp3	- (0,3 Sekunden)
pb15.mp3	Pistenbeleuchtung wird eingeschaltet, Leuchtdauer 1 – 5 Minuten

pb30.mp3	Pistenbeleuchtung wird eingeschaltet, Leuchtdauer 3 – 0 Minuten
pba1.mp3	Achtung: Pistenbeleuchtung wird in einer Minute ausgeschaltet, zur Reaktivierung 3 kurze Funkklicks senden.
pbas.mp3	Achtung: Pistenbeleuchtung wird ausgeschaltet, zur Reaktivierung 3 kurze Funkklicks senden.
qnh.mp3	Q – N – H
rc.mp3	Achtung: Modellflugbetrieb auf dem Fluggelände
s.mp3	südlichen Richtungen
schw.mp3	schwacher Wind aus
serr.mp3	zurzeit kann der Wind nicht angesagt werden
so.mp3	südöstlichen Richtungen
still.mp3	der Wind ist still
stkw.mp3	starker Wind aus
sw.mp3	südwestlichen Richtungen
w.mp3	westlichen Richtungen
wakt.mp3	Wetteransagesystem Michelstadt wurde aktiviert
wdok.mp3	Wetterdaten vom Sensor empfangen
wind.mp3	Wind aus
wndm.mp3	Wind mit

Ansage Format

Einleitung:	Michelstadt Radio
Wind:	leichter Wind aus südwest
Hinweis 1:	Die Piste ist zu überfliegen, dabei Wind und Hindernisfreiheit überprüfen!
Hinweis MFG:	Achtung: Modellflugbetrieb!
Hinweis KFZ:	Achtung: Kraftfahrzeugtestbetrieb auf dem Fluggelände!

Benutzerhandbuch

Funktionen

- Ansage von Windstärke, Windrichtung und QNH nach dem Erstanruf eines Piloten (Funkspruch mit mindestens 5 Sekunden Länge, danach erfolgen für 3 Minuten keine weiteren automatischen Wetteransagen).
- Die Uhrzeit (UTC), die Wetterdaten und der Zustand des Systems werden auf einer LCD-Anzeige (rechts neben dem Funkgerät) angezeigt.
- Steuerung der Pistenbeleuchtung mit entsprechenden Ansagen (Ansage, wenn die Beleuchtung eingeschaltet wird; Warnung eine Minute vor Abschalten der Beleuchtung; Ansage, wenn die Beleuchtung ausgeschaltet wird). Das Einschalten der Beleuchtung erfolgt durch das Senden von drei kurzen Funk-Klicks (Pausen zwischen den Klicks maximal 1,5 Sekunden). Die Beleuchtung kann durch das Senden von 5 Klicks auch wieder ausgeschaltet werden. Hierdurch wird gleichzeitig eine evtl. aktive, 3-minütige Stummschaltung der automatischen Wetteransage beendet.
- Durch einen Schlüsselschalter am Turm können zusätzliche Warnhinweise am Ende der Wetteransage aktiviert werden. Dies sind „KFZ-Testbetrieb“ und „Modellflugbetrieb“. Durch kurzes Drehen des Schlüssels (nach Osten = KFZ-Testbetrieb, nach Westen = Modellflug) werden die Warnungen aktiviert. Es kann jeweils nur eine der beiden Warnungen aktiviert werden. Wird der Schalter lange (> 3 Sekunden) in eine beliebige Richtung gedreht, so wird die Warnung abgeschaltet. Die Warnung schaltet sich nach vier Stunden Automatisch ab.
- Der Flugleiter kann die automatische Wetteransage temporär durch Drücken des schwarzen, runden Knopfes am Turmfunkgerät deaktivieren. Dann wird im Display ein „**X“** rechts neben der Uhrzeit angezeigt. Durch langes Drücken des Knopfes (> 3 Sekunden) wird die Stummschaltung beendet. Weiterhin kann der Flugleiter die Ansagen des Systems durch den Lautsprecher im Funkgerät mithören. Über das Satellitenfunkgerät des Flugleiters, können die automatischen Ansagen, aus technischen Gründen, leider nicht mitgehört werden. Um 18:00 Uhr UTC wird eine evtl. aktivierte Stummschaltung automatisch beendet. Ein anfliegender Pilot kann die Stummschaltung durch das Senden von 7 Klicks abschalten. Als Bestätigung der Abschaltung per Funk wird eine Wetteransage ausgesendet.

Display

W	I	N	D	:	3	6	0	°	1	0	k	t	s	PTT	M/P
Q	N	H	:	1	0	1	3		0	7	:	1	4	Stat.	B

Legende:

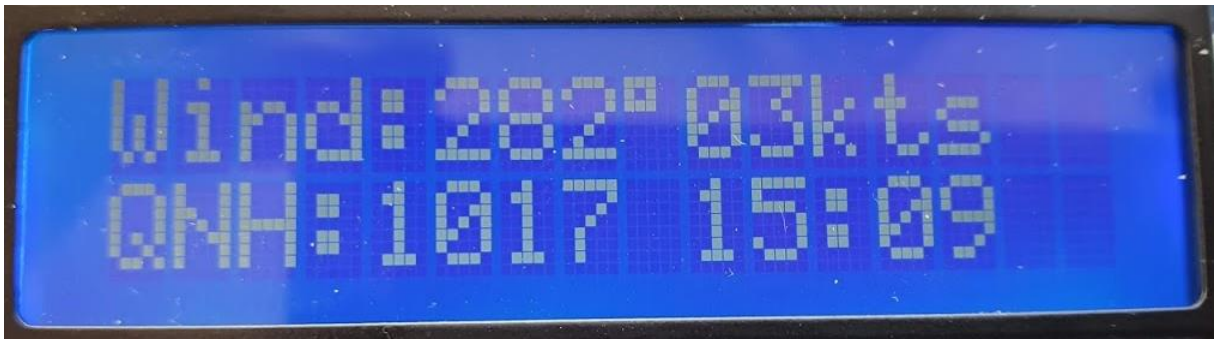
PPT ➔ „S“ = kurzer Funkklick erkannt (< 1 Sekunde)
„L“ = langer Funkklick erkannt (> 5 Sekunden)
„W“ = Wetteransage gestartet

M/P ➔ „M“ = Modellflug
„P“ = KFZ-Testbetrieb (Pirelli)

Stat. ➔ „X“ = Stummschaltung

B ➔ „B“ = Pistenbeleuchtung eingeschaltet

Beispiel:



Einstellung Funkgerät

Squelch-Einstellung

Um eine einwandfreie Funktion des Systems zu gewährleisten muss die Rauschsperre (Squelch) auf 20 eingestellt werden.

Mikrofonempfindlichkeit

Die Empfindlichkeit des dynamischen Mikrofoneinganges (DYN MIKE SENS) ist auf 22dB / 2.0 mV einzustellen.