```
/*
Modified to perform full scale detection and reporting by Glen Popiel -
(kw5gp [at] hotmail.com)

Based on LightningDetector.pde - AS3935 Franklin Lightning Sensor™ IC by
AMS library demo code
Copyright (c) 2012 Raivis Rengelis (raivis [at] rrkb.lv). All rights
reserved.
Modified in 2013 for I2C by Luka Mustafa - Musti (musti [at] wlan-
si.net).

This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 3 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
*/

// I2c library by Wayne Truchsess

#define debug  1 // Enables full diagnostic reporting

// I2C and AS3935 Libraries
#include "I2C.h"  // Use the I2C Library
#include <AS3935.h>  // Use the AS4935 Library
#include <LCD5110_Basic.h>  // Use the Nokia 5110 LCD Library
#include <Wire.h>  // Include the Wire Communication Library

/*
 It is assumed that the LCD module is connected to
 the following pins.
      CLK  - Pin 12
      DIN - Pin 11
      DC   - Pin 10
      RST  - Pin 8
      CE   - Pin 9
*/

LCD5110 glcd(12,11,10,8,9); //  Assign the Nokia 5110 LCD Pins

extern uint8_t SmallFont[];  // define the Nokia Font

// Lightning Detector library object initialization First argument is
interrupt pin, second is device I2C address
AS3935 AS3935(2,3);
```

```
#define IRQ_PIN 2  // Define Pin 2 for the Interrupt from the Detector
Module
#define SIMULATE_PIN 7  // Define Pin 7 for the Lightning Simulation
switch
#define holddown_time 20  // Delay to allow Detector Module settling
after startup
#define AS3935_ENERGY_HIGH 0x6,0x1f  // Defines the Register for the High
order bits of the lightning energy
#define AS3935_ENERGY_MID 0x5,0xff  // Defines the Register for the
Middle order bits of the lightning energy
#define AS3935_ENERGY_LOW 0x4,0xff  // Defines the Register for the Low
order bits of the lightning energy

#define NoiseFloor 2  // Define the Noise Floor Level of the Detector
Chip
#define SpikeReject 2  // Define the Spike Rejection Level of the
Detector Chip
#define Watchdog 2  // Define the Watchdog Setting of the Detector Chip

int recommended_tune_cap=5;  // Set the Recommended Value of the Detector
Tuning Capacitor
int test;  // calculated tuning cap value
int strokeIntensity;  // The intensity of the strike
int simulated;  // Indicates if strike is simulated
int irqSource;  // The source of the AS3935 interrupt
int strokeDistance;  // the distance of the strike
int holddown = 1;  // Set the Flag indicating startup
long last_event = 0;  // Holds the time of the last event detected
long last_tick =0;  // the time of the last minute "tick" in millis()
long current_time;  // the current time in millis()
long time;  // the difference in millis() between last "tick" and the
last event
// Used to calcluate the intensity of the strike
long strokeEnergyHigh, strokeEnergyMid, strokeEnergyLow, strokeTotal;
// The lines of text for the Nokia display
String text1, text2, text3, text4, text5, text6 = " ";

void setup()
{
  Serial.begin(9600);  // set the Serial USB port speed

  //I2C library initialization
  I2c.begin();
  I2c.pullup(true);
  I2c.setSpeed(0); //100kHz

  pinMode(IRQ_PIN, INPUT);  // Setup the Detector IRQ pin
  pinMode(SIMULATE_PIN, INPUT); // Setup the Lightning Simulate Button
  digitalWrite(SIMULATE_PIN, HIGH); // enable the pullup resistor on the
Simulate pin
  randomSeed(analogRead(0)); // seed the random number generator
  simulated = LOW; // reset the simulate flag
```

```
  // Set up the Nokia 5110 Display
  glcd.InitLCD(65);  // Initialize
  glcd.setFont(SmallFont);  // Use Small Font

  cleartext();  // clear the text values
  text1 = "KW5GP";
  text2 = "Lightning";
  text3 = "Detector";
  text6 = "Initializing";
  updatelcd();  // update the LCD

  if(debug == 1)   // If the debug flag is set, provide extra diagnostic
information
  {
    Serial.println("Scan I2C Bus");  // verify that we can see the
Lightning Detector (should be at 0x3)
    I2c.scan();  // Run the I2c Bus Scan function
  }



  Serial.println("Reset Detector");

  // reset all internal register values to defaults
  AS3935.reset();  // Reset the AS3935
  delay(1000);  // Wait a second for things to settle

// Set the Noise Floor, Spike Rejection and Watchdog settings

  AS3935.setNoiseFloor(NoiseFloor);  // Set the Noise Floor Level
  delay(200);
  AS3935.setSpikeRejection(SpikeReject);  // Set the Spike Rejection
Level
  delay(200);
  AS3935.setWatchdogThreshold(Watchdog);  // Set the Watchdog Level
  delay(1000);


  // The Embedded Adventures MOD-1016 includes the recommended Tuning
Capacitor Setting on the package
  // This Debug calibration routine is for testing and verification
purposes

  if (debug == 1)  // run calibration if debug flag set
  {
    // if lightning detector can not tune tank circuit to required
tolerance,
    // calibration function will return false
    if(!AS3935.calibrate()) {
      Serial.println("Tune Error");
    }
    delay(500);  // Wait for things to settle
    test = AS3935.registerRead(AS3935_TUN_CAP);   // read and display the
current Tuning Cap value
```

```
    delay(500);
    Serial.print("Tuning Cap: ");
    Serial.println(test,HEX);

  }
  // Set the Tuning Cap register to the value recommended
  Serial.println("Set Tune Cap Reg");
  AS3935.registerWrite(AS3935_TUN_CAP,recommended_tune_cap);  // Write
the recommended value to the Tuning Capacitor Register
  delay(500);  // Wait for things to settle
  test = AS3935.registerRead(AS3935_TUN_CAP);  // verify it is set
correctly
  delay(500);
  Serial.print("Tuning Cap: ");
  Serial.println(test,HEX);

  // uncomment below to use an oscilloscope to see LCO and SRCO signals
on IRQ line for 10 seconds each
  /*
  Serial.println("Set Disp LCO Reg 8 Bit 7");
  AS3935.registerWrite(AS3935_DISP_LCO,1);
  delay(10000);
  AS3935.registerWrite(AS3935_DISP_LCO,0);
  Serial.println("LCO disp complete");

  Serial.println("Set Disp SRCO Reg 8 Bit 6");
  AS3935.registerWrite(AS3935_DISP_SRCO,1);
  delay(10000);
  AS3935.registerWrite(AS3935_DISP_SRCO,0);
  Serial.println("SRCO disp complete");
  */

  // tell AS3935 we are indoors, for outdoors use setOutdoors() function
  AS3935.setIndoors();  // Set Gain for Indoors
  // AS3935.SetOutdoors;  // uncomment to set Gain for Outdoors
  delay(200);  // Wait for things to Settle

  //AS3935.enableDisturbers();  // Uncomment to turn on Disturber
Interrupts (EMI)
  AS3935.disableDisturbers();  // We only want Lightning, turn off EMI
interrupts
  delay(200);  // Wait for it things to settle
  printAS3935Registers();  // Display the registers

  int irqSource = AS3935.interruptSource(); // clear the IRQ before we
start
  delay(500);

  Serial.println("Detector Active"); // And we're ready for lightning

  cleartext();  // Clear all the LCD text variables
  text3 = "No Activity";
  updatelcd();  // Update the LCD
```

```
}  // End Setup Loop

void loop()    // Start the Main Loop
{
  // Check and update timestamp
  current_time = abs(millis())/1000;  // Current time (seconds since
start)
  if (current_time - last_tick >= 60) // Run if 60 seconds has passed
  {
    // One Minute has passed
    Serial.print("Tick  ");
    Serial.print(current_time);  // Print the current time
    Serial.print("   ");
    Serial.print(last_tick);  // Print the previous time
    Serial.print("   ");
    last_tick = current_time;
    // convert to minutes and hours
    time = last_tick - last_event; // convert difference last event to
current time into seconds
    Serial.print(time);

    if (time >=60 ) // One minute has passed
    {
      time = time/60;
      text6 = String(time);
      if ((time >= 1) && (time <60))
      // 1 to 59 minutes ago
      {
        text6 = text6 + " min ago";
      } else {
      if (time >=60)
      {
        time = time/60;
        // convert to hours
        text6 = String(time);

        if (time >=1)
        {
          if (time == 1)
          {
            text6 = text6 + " Hour ago";
          } else {
            text6 = text6 + " Hours ago";
          }
        }
      }
    }

    updatelcd();  // Update the LCD with time since last event
    }
    Serial.println("    " + text6);
  }
```

```
  if (holddown == 1)   // Delay the start for a few seconds to let
everything settle
  {
    if ((millis()/1000) > holddown_time)  // If we've passed the hold
down time
    {
      holddown = 0;  // Turn it loose

    }
  } else {

    // We've passed holddown time - rock and roll

    if (digitalRead(SIMULATE_PIN) == LOW) // Check for a simulation
    {
      simulated = HIGH;  // Turn on the simulated flag
      delay(1000); // disable the simulate button for 1 second
      Serial.print("Sim Button ");
    } else {
      simulated = LOW;  // Make sure we turn off the simulated flag
    }

    if ((digitalRead(IRQ_PIN) == HIGH) || (simulated))  // If we have a
real or simulated event let's do it
    {
      if (!simulated){       // if it's a real event, use the actual
interrupt, otherwise set the IRQ code for lightning
        delay(200);  // wait for interrupt register to settle
        irqSource = AS3935.interruptSource();  // Read the AS3935 IRQ
Register
        Serial.print("Real Irq ");
      } else {
        irqSource = 0b1000;  // Set the IRQ code for lighting
      }
      Serial.print("IRQ: ");
      Serial.print(irqSource);
      Serial.print(" ");
      // first step is to find out what caused interrupt
      // as soon as we read interrupt cause register, irq pin goes low
      // returned value is bitmap field, bit 0 - noise level too high,
bit 2 - disturber detected, and finally bit 3 - lightning!

      // create timestamp so we know when it occurred
      if(irqSource != 0)
      {  // 0 is a stat purge, we don't want to do anything with it
        timestamp();  // Run the timestamp function
        text6= "Now";  // Set the Text time of the event to "Now"
      }

      if (irqSource & 0b0001)  // Noise Level High Interrupt
      {
        text3 = "Noise High";
      }
```

```
      if (irqSource & 0b0100)  // Man Made Disturber (EMI) Interrupt
      {
        text3 = "EMI Detect";
      }

      if (irqSource & 0b1000)  // Lightning
      {
        // need to find distance of lightning strike, function returns
approximate distance in kilometers,
        // where value 1 represents storm in detector's near victinity,
and 63 - very distant, out of range stroke
        // everything in between is the distance in kilometers
        text2 = "Detected";
        if (simulated)
        {      // make up a distance if we're faking it
          strokeDistance = int(random(45));  // Pick a distance between 1
and 44)
          if (strokeDistance < 5 ) // If a real strike is less than 5km,
it's "Overhead", so we match that
          {
            strokeDistance = 1;
          }
          text1 = "Simulation";
        } else {    // otherwise, get the real distance
          strokeDistance = AS3935.lightningDistanceKm();  // It's real
lightning, read the AS3935 Distance Register
          text1 = "Lightning";
        }

        if (strokeDistance < 5) // The AS3935 Reports Lightning distance
as Out of Range,40,37,34,31,27,24,20,17,14,12,10,8,6,5,Overhead
        {

          text3 = "Overhead";
        }
        if (strokeDistance > 40)
        {
          text3 = "Out of Range";
        }
        if (strokeDistance <= 40 && strokeDistance >= 5)
        {
          text3 = String(strokeDistance) + " km away";
        }

        if (simulated)  // Make up the energy of the stroke
        {
          strokeEnergyHigh = int(random(31));  // There are 3 registers
containing strike energy
          strokeEnergyMid = int(random(255));
          strokeEnergyLow = int(random(255));
        } else {    // otherwise get the real energy
          strokeEnergyHigh = AS3935.registerRead(AS3935_ENERGY_HIGH);  //
Read the 3 Strike Energy Registers
          strokeEnergyMid = AS3935.registerRead(AS3935_ENERGY_MID);
```

```
          strokeEnergyLow = AS3935.registerRead(AS3935_ENERGY_LOW);
        }
        strokeTotal = strokeEnergyLow + (strokeEnergyMid*256)+
(strokeEnergyHigh*65536); // Calculate the total Strike Energy
        strokeIntensity = map(strokeTotal,1,2000000,1,10); //  map it to
an Intensity factor of 1 thru 10
        text4 = "Intensity: " + String(strokeIntensity);


      }
      if (irqSource == 0)
      {
        Serial.print(" Stats Purged ");  // The AS3935 will Purge old
lightning data automatically
      } else {

        updatelcd();  // Update the LCD with the Event Data
        Serial.print("Time: ");
        Serial.print(" ");
        Serial.print(last_event);
        Serial.print(" "+text1);
        //  Serial.print(text3);
        if (irqSource & 0b1000)
        {
          Serial.print("  Energy: ");
          Serial.print(String(strokeTotal));
          Serial.print("  Dist: ");
          Serial.print(strokeDistance);
          Serial.print(" ");
          Serial.print("Intensity: ");
          Serial.print(strokeIntensity);
          Serial.print(" ");
        }
      }
        Serial.print("Irq: ");
        Serial.print(irqSource);
        Serial.println("   " + text6);

    }
  }
}    // End the Main Loop

void printAS3935Registers()    // Display the basic AS3935 Registers
{
  int noiseFloor = AS3935.getNoiseFloor();  // Read the Noise Floor
setting
  int spikeRejection = AS3935.getSpikeRejection();  // Read the Spike
Rejection setting
  int watchdogThreshold = AS3935.getWatchdogThreshold();  // Read the
Watchdog Threshold setting
  Serial.print("Noise floor: ");
  Serial.println(noiseFloor,DEC);
  Serial.print("Spike reject: ");
  Serial.println(spikeRejection,DEC);
```

```
  Serial.print("WD threshold: ");
  Serial.println(watchdogThreshold,DEC);
}

void timestamp()      // stores the time of the last event in seconds
{
  last_event = abs(millis()/1000);
}

void updatelcd()    // clears LCD display and writes the current LCD text
data
{
  glcd.clrScr();
  glcd.print(text1,CENTER,0);
  glcd.print(text2,CENTER,8);
  glcd.print(text3,CENTER,16);
  glcd.print(text4,CENTER,24);
  glcd.print(text5,CENTER,32);
  glcd.print(text6,CENTER,40);
}

void cleartext()  // clears the text data
{
  text1 = " ";
  text2 = text1;
  text3 = text1;
  text4 = text1;
  text3 = text1;
  text4 = text1;
  text5 = text1;
  text6 = text1;
}
```