```
/*Sketch for the work of Arduino UNO and the ZERO II analyzer
without using a PC.
The analyzer operates in the frequency range from 100 kHz to
1000 MHz.
The operating frequency is set by rotating the encoder.
Measurement is started by pressing the encoder knob once.
One click of the encoder turn changes the frequency by 10 kHz,
one click of the turn with push of encoder knob changes the
frequency by 1 MHz.
Information is displayed on the LCD type 2004 I2C.
Used libraries:
  For display operation:
https://rigexpert.com/files/libraries/LiquidCrystal_I2C_V112/
  For the analyzer to work:
https://rigexpert.com/files/libraries/RigExpertZeroII_UART/
  The rest of the libraries are either standard or built into
the body of the sketch.
 This program code may be changed and used in any way without
notice to the author.
B© Alexander Antonov UT2UM June 05, 2022.*/


/* Added a library to work with the display. */
#include <Wire.h>
#include "LiquidCrystal_I2C.h"

/* Standard library for reading/writing data in EEPROM */
#include <EEPROM.h>

/* Display setting. Address I2C, number of columns, number of
rows. */
LiquidCrystal_I2C lcd(0x27, 20, 4);

/* Added a library to work with RigExpert ZERO II analyzer */
#include "RigExpertZeroII_UART.h"

/* ZERO pins */
#define RX_PIN 4
#define TX_PIN 7
#define ZEROII_Reset_Pin  6
RigExpertZeroII_UART ZERO(RX_PIN, TX_PIN);
```

```
/* The program code required for the encoder to work.
Do not change this program code unless necessary.
If you are not using an encoder, you can remove this code. */
class encMinim
{
  public:
    encMinim(uint8_t clk, uint8_t dt, uint8_t sw, boolean dir,
boolean type);
    void tick();
    boolean isClick();
    boolean isTurn();
    boolean isRight();
    boolean isLeft();
    boolean isRightH();
    boolean isLeftH();

  private:
    byte _clk, _dt, _sw;
    boolean _type = false;
    boolean _state, _lastState, _turnFlag, _swState, _swFlag,
_turnState;
    byte _encState;
    uint32_t _debTimer;
    // 0 - void, 1 - left, 2 - right, 3 - push_rigrt, 4 -
push_left
};

encMinim::encMinim(uint8_t clk, uint8_t dt, uint8_t sw, boolean
dir, boolean type) {
  if (dir) {
    _clk = clk;
    _dt = dt;
  } else {
    _clk = dt;
    _dt = clk;
  }
  _sw = sw;
  _type = type;
  pinMode (_clk, INPUT);
  pinMode (_dt, INPUT);
  pinMode (_sw, INPUT_PULLUP);
  _lastState = bitRead(PIND, _clk);
```

```
}

void encMinim::tick() {
  _encState = 0;
  _state = bitRead(PIND, _clk);
  _swState = bitRead(PIND, _sw);

  if (_state != _lastState) {
    _turnState = true;
    _turnFlag = !_turnFlag;
    if (_turnFlag || !_type) {
      if (bitRead(PIND, _dt) != _lastState) {
        if (_swState) _encState = 1;
        else _encState = 3;
      } else {
        if (_swState) _encState = 2;
        else _encState = 4;
      }
    }
    _lastState = _state;
  }

  if (!_swState && !_swFlag && millis() - _debTimer > 80) {
    _debTimer = millis();
    _swFlag = true;
    _turnState = false;
  }
  if (_swState && _swFlag && millis() - _debTimer > 80) {
    _debTimer = millis();
    _swFlag = false;
    if (!_turnState) _encState = 5;
  }
}
boolean encMinim::isTurn() {
  if (_encState > 0 && _encState < 5) {
    return true;
  } else return false;
}
boolean encMinim::isRight() {
  if (_encState == 1) {
    _encState = 0;
    return true;
```

```
  } else return false;
}
boolean encMinim::isLeft() {
  if (_encState == 2) {
    _encState = 0;
    return true;
  } else return false;
}
boolean encMinim::isRightH() {
  if (_encState == 3) {
    _encState = 0;
    return true;
  } else return false;
}
boolean encMinim::isLeftH() {
  if (_encState == 4) {
    _encState = 0;
    return true;
  } else return false;
}
boolean encMinim::isClick() {
  if (_encState == 5) {
    _encState = 0;
    return true;
  } else return false;
}
/* End code */

/* encoder pins */
#define SW 5
#define DT 3
#define CLK 2

/* For encoder: pin CLK, pin DT, pin SW, direction of rotation
(0/1), type (0/1) */
encMinim enc(CLK, DT, SW, 1, 0);

/* Variable for storing the value of the operating frequency, Hz
*/
volatile int32_t freq = 0;
```

```
/* variable "timer" so as not to use the standard delay function
*/
int32_t timer = millis();

void setup()
{
  EEPROM.get(0, freq);
                          // Reading the frequency value from ROM
  delay(50);
                          // required delay
  lcd.init();
                          // Initialize display
  delay(10);
                          // required delay
  lcd.backlight();
                          // Turn on display backlight
  pinMode(ZEROII_Reset_Pin, OUTPUT);
                    // Reset analyzer
  digitalWrite(ZEROII_Reset_Pin, LOW);
  delay(300);
  digitalWrite(ZEROII_Reset_Pin, HIGH );

/* Check if the analyzer is found? */
  while (!ZERO.startZeroII())
  {
    lcd.setCursor(6, 0);
                          // Set the cursor
    lcd.print("Analyzer");
                          // and print text on the display
    lcd.setCursor(5, 1);
    lcd.print("not found.");
    lcd.setCursor(4, 3);
    delay(1000);
    lcd.print("rechecking...");
    delay(1000);
                          // recheck after 1 second
  }

/* If found, then display the greeting on the screen. */
  lcd.setCursor(1, 0);
  lcd.print("RigExpert ZERO II");
  lcd.setCursor(6, 1);
```

```
  lcd.print("analyzer");
  lcd.setCursor(0, 3);
  lcd.print("FW:");
                                        // firmware version display
  lcd.setCursor(3, 3);
  lcd.print(ZERO.getMajorVersion());
  lcd.setCursor(4, 3);
  lcd.print(".");
  lcd.setCursor(5, 3);
  lcd.print(ZERO.getMinorVersion());
  lcd.setCursor(8, 3);
  lcd.print("S/N");
  lcd.setCursor(11, 3);
  lcd.print(ZERO.getSerialNumber());
                    // firmware version display
  delay(3000);
                          // Display splash screen 3 seconds
  lcd.clear();
                          // Clear display
  lcd.setCursor(2, 2);
  lcd.print("press to measure");
                          // The analyzer is ready to work
}

void loop()
{
  delay(10);
  enc.tick();
                    // Initialize rotary encoder
  delay(10);

/* Rotary encoder processing */
  if (enc.isLeft())
  {
    freq_left();
  }
  if (enc.isRight())
  {
    freq_right();
  }
  if (enc.isLeftH())
  {
```

```
  freq_leftH();
}
if (enc.isRightH())
{
  freq_rightH();
}
if (enc.isClick())
{
  start_m ();
}

/* Update the frequency value on screen every 200 ms. */
if (millis() - timer > 200)
{
  if ((freq / 1000) >= 100000)
  {
    lcd.setCursor(0, 0);
    lcd.print("Freq = ");
    lcd.setCursor(13, 0);
    lcd.print(' ');
    lcd.setCursor(14, 0);
    lcd.print(' ');
    lcd.setCursor(7, 0);
    float f_ = (freq / 1000000.0);
    lcd.print(f_);
    lcd.setCursor(17, 0);
    lcd.print("MHz");
    timer = millis();
  }
  if ((freq / 1000.0) < 99999.0 & (freq / 1000.0) >= 10000.0)
  {
    lcd.setCursor(0, 0);
    lcd.print("Freq = ");
    lcd.setCursor(10, 0);
    lcd.print(' ');
    lcd.setCursor(7, 0);
    float f_ = (freq / 1000.0);
    lcd.print(f_);
    lcd.setCursor(17, 0);
    lcd.print("kHz");
    timer = millis();
  }
```

```
    if ((freq / 1000.0) < 9999.0 & (freq / 1000.0) >= 1000.0)
    {
      lcd.setCursor(0, 0);
      lcd.print("Freq = ");
      lcd.setCursor(14, 0);
      lcd.print(' ');
      lcd.setCursor(15, 0);
      lcd.print(' ');
      lcd.setCursor(16, 0);
      lcd.print(' ');
      lcd.setCursor(7, 0);
      float f_ = (freq / 1000.0);
      lcd.print(f_);
      lcd.setCursor(17, 0);
      lcd.print("kHz");
      timer = millis();
    }
      if ((freq / 1000.0) < 999.0)
      {
      lcd.setCursor(0, 0);
      lcd.print("Freq = ");
      lcd.setCursor(13, 0);
      lcd.print(' ');
      lcd.setCursor(14, 0);
      lcd.print(' ');
      lcd.setCursor(15, 0);
      lcd.print(' ');
      lcd.setCursor(16, 0);
      lcd.print(' ');
      lcd.setCursor(7, 0);
      float f_ = (freq / 1000.0);
      lcd.print(f_);
      lcd.setCursor(17, 0);
      lcd.print("kHz");
      timer = millis();
    }
  }
}

void freq_leftH ()
{
  freq = freq + 1000000;
```

```
  if (freq >= 1000000000)
  {
    freq = 1000000000;
  }
  save_freq();
}

void freq_rightH ()
{
  freq = freq - 1000000;
  if (freq <= 100000)
  {
    freq = 100000;
  }
  save_freq();
}

void freq_left ()
{
  freq = freq + 10000;
  if (freq >= 1000000000)
  {
    freq = 1000000000;
  }
  save_freq();
}

void freq_right ()
{
  freq = freq - 10000;
  if (freq <= 100000)
  {
    freq = 100000;
  }
  save_freq();
}

inline void save_freq()
{
                    // Save the current frequency value to ROM
  EEPROM.put(0, freq);
}
```

```
void start_m ()
{
  ZERO.startZeroII();
  delay(10);
  ZERO.startMeasure(freq);
                    // start measurement
  //delay(100);
  lcd.clear();
  lcd.setCursor(0, 1);
  lcd.print("SWR");
  float SWR = ZERO.getSWR();
            // get SWR value
  lcd.setCursor(4, 1);
  if (SWR < 10)
  {
    lcd.print(SWR, 2);
  }
  if ((SWR >= 10) && (SWR < 100))
  {
    lcd.print(SWR, 1);
  }
  if ((SWR >= 100) && (SWR <= 200))
  {
    lcd.print(">100");
  }
  float Z = ZERO.getZ();
            // get Z value
  lcd.setCursor(11, 1);
  lcd.print('Z');
  lcd.setCursor(13, 1);
  lcd.print(Z, 2);
  lcd.setCursor(0, 2);
  lcd.print("R");
  float R = ZERO.getR();
                  // get R value
  lcd.setCursor(4, 2);
  lcd.print(R, 1);
  float X = ZERO.getX();
            // get X value
  lcd.setCursor(11, 2);
  lcd.print('X');
```

```
lcd.setCursor(13, 2);
if (X <= 999.0)
{
  lcd.print(X, 1);
}
if (X > 1000.0)
{
  lcd.print(">1000");
}
lcd.setCursor(0, 3);
lcd.print("RL");
float RL = ZERO.getRL();
                // get RL value
lcd.setCursor(4, 3);
lcd.print(RL, 2);
float Rho = ZERO.getRho();
          // get Rho value
lcd.setCursor(11, 3);
lcd.print("Rho");
lcd.setCursor(15, 3);
lcd.print(Rho);
```