```
// 20240220_OLEDoscilloScope_on_R909
// Simple oscillo scope on R909-PANEL by nobcha, kpa radio
//  OLED:SSD1306, Rotary encoder & push switch
//  Port assign changed
//    INPUT:A3, ATT:D4, RE:D2&D3 , RE-SW:A0 , FUNCTION SW (SW1&2): A2
//    debug monitor on L763
//

// Many thanks RADIO_PENCH for providing cool device
// Simple oscillo scope (20200726_OLEDoscilloscopeSh1106_V300E.ino)
/*
  PMO-RP1 V3.0 added AC range, chnged the others
  20200726_OLEDoscilloscopeSh1106_V300E.ino
  SH1106 sketch:24246byte, local variable:209byte free
  SD1306 sketch:26006byte,local:1223byte
  xxx.xxx 2020 by radiopench http://radiopench.blog96.fc2.com/
*/

#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
//#include <Adafruit_SH1106.h>          //
https://github.com/wonho-maker/Adafruit_SH1106
#include <EEPROM.h>
#include <Rotary.h>                     //Ben  Buxton
https://github.com/brianlow/Rotary

#define SCREEN_WIDTH   128              // OLED display width
#define SCREEN_HEIGHT   64              // OLED display height
#define REC_LENG       200              // size of wave data buffer
#define MIN_TRIG_SWING   5              // minimum trigger swing.(Display "Unsync" if
swing smaller than this value

// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
#define OLED_RESET     -1               // Reset pin # (or -1 if sharing Arduino reset
pin)
Adafruit_SSD1306 oled(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);   // device name
is oled
//Adafruit_SH1106 oled(OLED_RESET);        // use this when SH1106

#define REA 2             // Rotary encoder A node
#define REB 3             // Rotary encoder B node
#define INPUT_PORT A3     //
#define RE_SW A0          // Select button
#define FUNC_SW A2        // SW1 SW2
#define LED_PORT A1       // LED
#define R_12k   4         // 12kohm on D12
#define R_820k  5         // 820k ohm  for AC low range
#define R_82k   6         // 82k omm for AC Hi range
#define AC_DC   7

#define HALF  // better

Rotary  r = Rotary(REA, REB);
unsigned char re_result = 3;
ISR(PCINT2_vect) {
  re_result = r.process();
}

// Range name table (those are stored in flash memory)
const char vRangeName[10][5] PROGMEM = {"A50V", "A 5V", " 50V", " 20V", " 10V", "  5V",
"  2V", "  1V", "0.5V", "0.2V"}; // Vertical display character (number of characters
including ¥ 0 is required)
const char * const vstring_table[] PROGMEM = {vRangeName[0], vRangeName[1],
```

```
vRangeName[2], vRangeName[3], vRangeName[4], vRangeName[5], vRangeName[6],
vRangeName[7], vRangeName[8], vRangeName[9]};
const char hRangeName[12][6] PROGMEM = {"200ms", "100ms", " 50ms", " 20ms", " 10ms", "
5ms", "  2ms", "  1ms", "500us", "200us", "100us", " 50us"};  //  Hrizontal display
characters
const char * const hstring_table[] PROGMEM = {hRangeName[0], hRangeName[1],
hRangeName[2], hRangeName[3], hRangeName[4], hRangeName[5], hRangeName[6],
hRangeName[7], hRangeName[8], hRangeName[9], hRangeName[10], hRangeName[11]};
const PROGMEM float hRangeValue[] = { 0.2, 0.1, 0.05, 0.02, 0.01, 0.005, 0.002, 0.001,
0.5e-3, 0.2e-3, 0.2e-3, 0.2e-3}; // record speed in second. ( = 25pix on screen) this
value used for freq calc.

int waveBuff[REC_LENG];        // wave form buffer (RAM remaining capacity is barely)
char chrBuff[8];               // display string buffer
char hScale[] = "xxxAs";       // horizontal scale character
char vScale[] = "xxxx";        // vartical scale

float lsb5V    = 0.00563965;   // lsb5V(5V)sensivity coefficient of 5V range.
std=0.00563965 1.1*630/(1024*120)
float lsb50V   = 0.0512939;    // lsb50V(50V)sensivity coefficient of 50V range.
std=0.0512939 1.1*520.91/(1024*10.91)

float lsb5Vac  = 0.00630776;   // lsb5V(ac) voltage for 1LSB(V) std=0.00630776 V/LSB
float lsb50Vac = 0.0579751;    // lsb50V(50V) voltage for 1LSB(V) std=0.0579751 V/LSB

volatile int vRange;           // V-range number                    2:50V,  3:20V,
4:10V,  5:5V,  6:2V,  7:1V,  8:0.5V,  9:0.2V
volatile int hRange;           // H-range nubmer 0:200ms, 1:100ms, 2:50ms, 3:20ms,
4:10ms, 5:5ms, 6;2ms, 7:1ms, 8:500us, 9;200us
volatile int trigD;            // trigger slope flag,     0:positive 1:negative
volatile int scopeP;           // operation scope position number. 0:Veratical,
1:Hrizontal, 2:Trigger slope
volatile boolean hold = false; // hold flag
volatile boolean switchPushed = false; // flag of switch pusshed !
volatile int saveTimer;        // remaining time for saving EEPROM
int timeExec;                  // approx. execution time of current range setting (ms)

int dataMin;                   // buffer minimum value (smallest=0)
int dataMax;                   //        maximum value (largest=1023)
int dataAve;                   // 10 x average value (use 10x value to keep accuracy.
so, max=10230)
int dataRms;                   // 10x rms. value
int rangeMax;                  // buffer value to graph full swing
int rangeMin;                  // buffer value of graph botto
int rangeMaxDisp;              // display value of max. (100x value)
int rangeMinDisp;              // display value if min.
int trigP;                     // trigger position pointer on data buffer
boolean trigSync;              // flag of trigger detected
int att10x;                    // 10x attenetor ON (effective when 1)
int inMode;                    // 0=DC+, 1=DC+-, 2=AC
int offset5Vac;
int offset50Vac;

float waveFreq;                // frequency (Hz)
float waveDuty;                // duty ratio (%)

void setup() {
  pinMode(REA, INPUT_PULLUP);           // rotary encoder A
  pinMode(REB, INPUT_PULLUP);           // rotary encoder B
  pinMode(RE_SW, INPUT_PULLUP);         // Select button
  pinMode(LED_PORT, OUTPUT);            // LED
  pinMode(R_12k, INPUT);                // pin D4 1/10 attenuator(Off=High-Z,
Enable=Output Low)
  pinMode(R_820k, INPUT);               // D5
```

```
  pinMode(R_82k, INPUT);                // D6
  pinMode(AC_DC, INPUT_PULLUP);         //

//  uuPinOutputLow(0b00000001111000,0b000000); // output low at pin D3-D6
  oled.begin(SSD1306_SWITCHCAPVCC, 0x3C); // select 3C or 3D (set your OLED I2C
address)
//  oled.begin(SH1106_SWITCHCAPVCC, 0x3C);  // use this when SH1106

  auxFunctions();                       // Voltage measure (never return)
  loadEEPROM();                         // read last settings from EEPROM
  analogReference(INTERNAL);            // ADC full scale = 1.1V

  r.begin();
  PCICR |= (1 << PCIE2);                // Set encoder interrupt
  PCMSK2 |= (1 << PCINT18) | (1 << PCINT19);  // Set encoder interrupt
  sei();                                // Enable interrupt

  startScreen();                        // display start message

}

void loop() {
  setInputOffset();                     // coupling mode set(AC/DC)
  setConditions();                      // set measurment conditions
  digitalWrite(LED_PORT, HIGH);         // flash LED
  readWave();                           // read wave form and store into buffer memory
  digitalWrite(LED_PORT, LOW);          // stop LED
  setConditions();                      // set measurment conditions again (reflect
change during measure)
  dataAnalize();                        // analize data
  writeCommonImage();                   // write fixed screen image (2.6ms)
  plotData();                           // plot waveform (10-18ms)
  dispInf();                            // display information (6.5-8.5ms)
  oled.display();                       // send screen buffer to OLED (37ms)
  saveEEPROM();                         // save settings to EEPROM if necessary
  key_sense();                          //
  while (analogRead( FUNC_SW ) <400 ) { // wait if Hold SW ON
    dispHold();
    if (inMode > 0) {                   // if DC mode,
      if (acZero() == 1) {              // if offset adj. executed
        scopeP = 0;                     // scope position to vartical
        hold = false;                   // cancel hold
      }
      delay(10);
    }                                   //
  }
}

int acZero() {                          // cancel AC renge offset
  if (digitalRead(RE_SW) == LOW) {      // if select pushed
    if (vRange >= 5) {                  // range = 5V or less
      offset5Vac = dataAve / 10;        // adjust the offset
    } else {                            // range more than 5V
      offset50Vac = dataAve / 10;       // adjust the offset
    }
    saveEEPROM();                       // EEPROM
    return 1;                           //  adjusted
  }
  return 0;                             //  no adjust
}

void setInputOffset() {                 // set offset circuit
  if (inMode >= 1) {                    // if AC mode
    if (att10x == 1) {                  // 10X-att enabled
```

```
      pull5V(R_82k);                    // pull 5V by 82k
      hiZ(R_820k);
    } else {                            // 10X-att disable
      hiZ(R_82k);
      pull5V(R_820k);                   // pull 5V by 820k
    }
  } else {                              // DC mode
    hiZ(R_820k);                        // Hi-Z
    hiZ(R_82k);                         // Hi-Z
  }
}

void hiZ(int n) {                       // set the pin to hi-z
  pinMode(n, INPUT);                    // set INPUT
  digitalWrite(n, LOW);                 // no pull up
}

void pull5V(int n) {                    // pull 5V through registor
  pinMode(n, OUTPUT);                   // set OUTPUT
  digitalWrite(n, HIGH);                // OUTPUT HIGH
}

void pullGND(int n) {                   // pull GND through registor
  pinMode(n, OUTPUT);                   // set OUTPUT
  digitalWrite(n, LOW);                 // output LOW
}

void setConditions() {                  //
  if (digitalRead(AC_DC) == LOW) {      // Aset AC/DC
    inMode = 1;                         // h
  } else {
    inMode = 0;                         // h
  }

  // get range name from PROGMEM
  strcpy_P(hScale, (char*)pgm_read_word(&(hstring_table[hRange])));  // H range name
  strcpy_P(vScale, (char*)pgm_read_word(&(vstring_table[vRange])));  // V range name

  switch (vRange) {                 // setting of Vrange
    case 0:                         // Auto50V range
      att10x = 1;                   // use input attenuator
      break;

    case 1:                         // Auto 5V range
      att10x = 0;                   // no attenuator
      break;

    case 2:                         // 50V range
      if (inMode == 0) {
        rangeMax = 50.0 / lsb50V;   // set full scale pixcel count number
        rangeMaxDisp = 5000;        // vartical scale (set100x value)
        rangeMin = 0;
        rangeMinDisp = 0;
      } else {
        rangeMax = offset50Vac + 25.0 / lsb50Vac; // set full scale pixcel count number
        rangeMaxDisp = 2500;        // vartical scale (set100x value)

        rangeMin = offset50Vac - 25.0 / lsb50Vac;
        rangeMinDisp = -2500;
      }
      att10x = 1;                   // use input attenuator
      break;

    case 3:                         // 20V range
```

```
      if (inMode == 0) {
        rangeMax = 20.0 / lsb50V;  // set full scale pixcel count number
        rangeMaxDisp = 2000;
        rangeMin = 0;
        rangeMinDisp = 0;
      } else {
        rangeMax = offset50Vac + 10.0 / lsb50Vac;  // set full scale pixcel count
number
        rangeMaxDisp = 1000;
        rangeMin = offset50Vac - 10.0 / lsb50Vac;
        rangeMinDisp = -1000;
      }
      att10x = 1;                  // use input attenuator
      break;

    case 4:                        // 10V range
      if (inMode == 0) {
        rangeMax = 10.0 / lsb50V;  // set full scale pixcel count number
        rangeMaxDisp = 1000;
        rangeMin = 0;
        rangeMinDisp = 0;
      } else {
        rangeMax = offset50Vac + 5.0 / lsb50Vac;  // set full scale pixcel count number
        rangeMaxDisp = 500;
        rangeMin = offset50Vac - 5.0 / lsb50Vac;
        rangeMinDisp = -500;
      }
      att10x = 1;                  // use input attenuator
      break;

    case 5:                        // 5V range
      if (inMode == 0) {
        rangeMax = 5.0 / lsb5V;    // set full scale pixcel count number
        rangeMaxDisp = 500;
        rangeMin = 0;
        rangeMinDisp = 0;
      } else {
        rangeMax = offset5Vac + 2.5 / lsb5Vac;    // set full scale pixcel count number
        rangeMaxDisp = 250;
        rangeMin = offset5Vac - 2.5 / lsb5Vac;
        rangeMinDisp = -250;
      }
      att10x = 0;                  // no input attenuator
      break;

    case 6:                        // 2V range
      if (inMode == 0) {
        rangeMax = 2.0 / lsb5V;    // set full scale pixcel count number
        rangeMaxDisp = 200;
        rangeMin = 0;
        rangeMinDisp = 0;
      } else {
        rangeMax = offset5Vac + 1.0 / lsb5Vac;    // set full scale pixcel count number
        rangeMaxDisp = 100;
        rangeMin = offset5Vac - 1.0 / lsb5Vac;
        rangeMinDisp = -100;
      }
      att10x = 0;                  // no input attenuator
      break;

    case 7:                        // 1V range
      if (inMode == 0) {
        rangeMax = 1.0 / lsb5V;    // set full scale pixcel count number
        rangeMaxDisp = 100;
```

```
        rangeMin = 0;
        rangeMinDisp = 0;
      } else {
        rangeMax = offset5Vac + 0.5 / lsb5Vac;    // set full scale pixcel count number
        rangeMaxDisp = 50;
        rangeMin = offset5Vac - 0.5 / lsb5Vac;
        rangeMinDisp = -50;
      }
      att10x = 0;                  // no input attenuator
      break;

    case 8:                        // 0.5V range
      if (inMode == 0) {
        rangeMax = 0.5 / lsb5V;    // set full scale pixcel count number
        rangeMaxDisp = 50;
        rangeMin = 0;
        rangeMinDisp = 0;
      } else {
        rangeMax = offset5Vac + 0.25 / lsb5Vac;  // set full scale pixcel count number
        rangeMaxDisp = 25;
        rangeMin = offset5Vac - 0.25 / lsb5Vac;
        rangeMinDisp = -25;
      }
      att10x = 0;                  // no input attenuator
      break;

    case 9:                        // 0.2V range
      if (inMode == 0) {
        rangeMax = 0.2 / lsb5V;    // set full scale pixcel count number
        rangeMaxDisp = 20;
        rangeMin = 0;
        rangeMinDisp = 0;
      } else {
        rangeMax = offset5Vac + 0.1 / lsb5Vac;  // set full scale pixcel count number
        rangeMaxDisp = 10;
        rangeMin = offset5Vac - 0.1 / lsb5Vac;
        rangeMinDisp = -10;
      }
      att10x = 0;                  // no input attenuator
      break;
  }
}

void writeCommonImage() {                // Common screen image drawing
  oled.clearDisplay();                   // erase all(0.4ms)
  oled.setTextColor(WHITE);              // write in white character
  oled.drawFastVLine(26, 9, 55, WHITE);  // left vartical line
  oled.drawFastVLine(127, 9, 3, WHITE);  // right vrtical line up
  oled.drawFastVLine(127, 61, 3, WHITE); // right vrtical line bottom

  oled.drawFastHLine(24, 9, 7, WHITE);   // Max value auxiliary mark
  oled.drawFastHLine(24, 36, 2, WHITE);
  oled.drawFastHLine(24, 63, 7, WHITE);

  oled.drawFastHLine(51, 9, 3, WHITE);   // Max value auxiliary mark
  oled.drawFastHLine(51, 63, 3, WHITE);

  oled.drawFastHLine(76, 9, 3, WHITE);   // Max value auxiliary mark
  oled.drawFastHLine(76, 63, 3, WHITE);

  oled.drawFastHLine(101, 9, 3, WHITE);  // Max value auxiliary mark
  oled.drawFastHLine(101, 63, 3, WHITE);

  oled.drawFastHLine(123, 9, 5, WHITE);  // right side Max value auxiliary mark
```

```
  oled.drawFastHLine(123, 63, 5, WHITE);

  for (int x = 26; x <= 128; x += 5) {
    oled.drawFastHLine(x, 36, 2, WHITE);  // Draw the center line (horizontal line)
with a dotted line
  }
  for (int x = (127 - 25); x > 30; x -= 25) {
    for (int y = 10; y < 63; y += 5) {
      oled.drawFastVLine(x, y, 2, WHITE); // Draw 3 vertical lines with dotted lines
    }
  }
}

void readWave() {                         // Record waveform to memory array
  if (att10x == 1) {                      // if 1/10 attenuator required
    pullGND(R_12k);
  } else {                                // if not required
    hiZ(R_12k);
  }
  switchPushed = false;                   // Clear switch operation flag

  switch (hRange) {                       // set recording conditions in accordance
with the range number
    case 0:                               // 200ms range
      timeExec = 1600 + 60;               // Approximate execution time(ms) Used for
countdown until saving to EEPROM
      ADCSRA = ADCSRA & 0xf8;             // clear bottom 3bit
      ADCSRA = ADCSRA | 0x07;             // dividing ratio = 128 (default of Arduino
j
      for (int i = 0; i < REC_LENG; i++) { // up to rec buffer size
        waveBuff[i] = analogRead(INPUT_PORT);       // read and save approx 112us
        delayMicroseconds(7888);          // timing adjustment
        if (switchPushed == true) {       // if any switch touched
          switchPushed = false;
          break;                          // abandon record(this improve response)
        }
      }
      break;

    case 1:                               // 100ms range
      timeExec = 800 + 60;                // Approximate execution time(ms) Used for
countdown until saving to EEPROM
      ADCSRA = ADCSRA & 0xf8;             // clear bottom 3bit
      ADCSRA = ADCSRA | 0x07;             // dividing ratio = 128 (default of Arduino
j
      for (int i = 0; i < REC_LENG; i++) { // up to rec buffer size
        waveBuff[i] = analogRead(INPUT_PORT);       // read and save approx 112us
        delayMicroseconds(3860);          // timing adjustmet tuned
        if (switchPushed == true) {       // if any switch touched
          switchPushed = false;
          break;                          // abandon record(this improve response)
        }
      }
      break;

    case 2:                               // 50ms range
      timeExec = 400 + 60;                // Approximate execution time(ms)
      ADCSRA = ADCSRA & 0xf8;             // clear bottom 3bit
      ADCSRA = ADCSRA | 0x07;             // dividing ratio = 128 (default of Arduino
j
      for (int i = 0; i < REC_LENG; i++) { // up to rec buffer size
        waveBuff[i] = analogRead(INPUT_PORT);       // read and save approx 112us
        delayMicroseconds(1880);          // timing adjustmet tuned
        if (switchPushed == true) {       // if any switch touched
```

```
          break;                            // abandon record(this improve response)
        }
      }
      break;

    case 3:                                 // 20ms range
      timeExec = 160 + 60;                  // Approximate execution time(ms)
      ADCSRA = ADCSRA & 0xf8;               // clear bottom 3bit
      ADCSRA = ADCSRA | 0x07;               // dividing ratio = 128 (default of Arduino
j
      for (int i = 0; i < REC_LENG; i++) { // up to rec buffer size
        waveBuff[i] = analogRead(INPUT_PORT);       // read and save approx 112us
        delayMicroseconds(686);             // timing adjustmet tuned
        if (switchPushed == true) {         // if any switch touched
          break;                            // abandon record(this improve response)
        }
      }
      break;

    case 4:                                 // 10ms range
      timeExec = 80 + 60;                   // Approximate execution time(ms)
      ADCSRA = ADCSRA & 0xf8;               // clear bottom 3bit
      ADCSRA = ADCSRA | 0x07;               // dividing ratio = 128 (default of Arduino
j
      for (int i = 0; i < REC_LENG; i++) { // up to rec buffer size
        waveBuff[i] = analogRead(INPUT_PORT);       // read and save approx 112us
        delayMicroseconds(287);             // timing adjustmet tuned
        if (switchPushed == true) {         // if any switch touched
          break;                            // abandon record(this improve response)
        }
      }
      break;

    case 5:                                 // 5ms range
      timeExec = 40 + 60;                   // Approximate execution time(ms)
      ADCSRA = ADCSRA & 0xf8;               // clear bottom 3bit
      ADCSRA = ADCSRA | 0x07;               // dividing ratio = 128 (default of Arduino
j
      for (int i = 0; i < REC_LENG; i++) { // up to rec buffer size
        waveBuff[i] = analogRead(INPUT_PORT);       // read and save approx 112  s
        delayMicroseconds(87);              // timing adjustmet tuned
        if (switchPushed == true) {         // if any switch touched
          break;                            // abandon record(this improve response)
        }
      }
      break;

    case 6:                                 // 2ms range
      timeExec = 16 + 60;                   // Approximate execution time(ms)
      ADCSRA = ADCSRA & 0xf8;               // clear bottom 3bit
      ADCSRA = ADCSRA | 0x06;               // dividing ratio = 64 (0x1=2, 0x2=4, 0x3=8,
0x4=16, 0x5=32, 0x6=64, 0x7=128)
      for (int i = 0; i < REC_LENG; i++) { // up to rec buffer size
        waveBuff[i] = analogRead(INPUT_PORT);       // read and save approx 56us
        delayMicroseconds(23);              // timing adjustmet tuned
      }
      break;

    case 7:                                 // 1ms range
      timeExec = 8 + 60;                    // Approximate execution time(ms)
      ADCSRA = ADCSRA & 0xf8;               // clear bottom 3bit
      ADCSRA = ADCSRA | 0x05;               // dividing ratio = 16 (0x1=2, 0x2=4, 0x3=8,
0x4=16, 0x5=32, 0x6=64, 0x7=128)
      for (int i = 0; i < REC_LENG; i++) { // up to rec buffer size
```

```
        waveBuff[i] = analogRead(INPUT_PORT);       // read and save approx 28us
        delayMicroseconds(10);              // timing adjustmet tuned
      }
      break;

    case 8:                                 // 500us range
      timeExec = 4 + 60;                    // Approximate execution time(ms)
      ADCSRA = ADCSRA & 0xf8;               // clear bottom 3bit
      ADCSRA = ADCSRA | 0x04;               // dividing ratio = 16(0x1=2, 0x2=4, 0x3=8,
0x4=16, 0x5=32, 0x6=64, 0x7=128)
      for (int i = 0; i < REC_LENG; i++) { // up to rec buffer size
        waveBuff[i] = analogRead(INPUT_PORT);       // read and save approx 16us
        delayMicroseconds(4);               // timing adjustmet
        // time fine adjustment 0.0625 x 8 = 0.5us inop=0.0625us @16MHz)
        asm("nop"); asm("nop"); asm("nop"); asm("nop"); asm("nop"); asm("nop");
asm("nop"); asm("nop");
      }
      break;

    case 9:
    case 10:
    case 11:                                // common 200, 100, 50us range
      timeExec = 2 + 60;                    // Approximate execution time(ms)
      ADCSRA = ADCSRA & 0xf8;               // clear bottom 3bit
      ADCSRA = ADCSRA | 0x02;               // dividing ratio = 4(0x1=2, 0x2=4, 0x3=8,
0x4=16, 0x5=32, 0x6=64, 0x7=128)
      for (int i = 0; i < REC_LENG; i++) { // up to rec buffer size
        waveBuff[i] = analogRead(INPUT_PORT);       // read and save approx 6us
        // time fine adjustment 0.0625 * 20 = 1.25us (nop=0.0625us @16MHz)
        asm("nop"); asm("nop"); asm("nop"); asm("nop"); asm("nop"); asm("nop");
asm("nop"); asm("nop"); asm("nop"); asm("nop");
        asm("nop"); asm("nop"); asm("nop"); asm("nop"); asm("nop"); asm("nop");
asm("nop"); asm("nop"); asm("nop"); asm("nop");
      }
      break;
  }
}

void dataAnalize() {                        // get various information from wave form
  long d;
  long sum = 0;

  // search max and min value
  dataMin = 1023;                           // min value initialize to big number
  dataMax = 0;                              // max value initialize to small number
  for (int i = 0; i < REC_LENG; i++) {      // serach max min value
    d = waveBuff[i];
    sum = sum + d;
    if (d < dataMin) {                      // update min
      dataMin = d;
    }
    if (d > dataMax) {                      // updata max
      dataMax = d;
    }
  }

  // calculate average
  dataAve = (sum + 10) / 20;                // Average value calculation (calculated by
10 times to improve accuracy)

  // rms value calc.
  sum = 0;
  for (int i = 0; i < REC_LENG; i++) {      // to all buffer
    d = waveBuff[i] - (dataAve + 5) / 10;  //
```

```
    sum += d * d;                           //
  }
  dataRms = sqrt(sum / REC_LENG);           // get rms value

  // Trigger position search
  for (trigP = ((REC_LENG / 2) - 51); trigP < ((REC_LENG / 2) + 50); trigP++) { // Find
the points that straddle the median at the center  } 50 of the data range
    if (trigD == 0) {                                // if trigger direction is positive
      if ((waveBuff[trigP - 1] < (dataMax + dataMin) / 2) && (waveBuff[trigP] >=
(dataMax + dataMin) / 2)) {
        break;                                       // positive trigger position found !
      }
    } else {                                         // trigger direction is negative
      if ((waveBuff[trigP - 1] > (dataMax + dataMin) / 2) && (waveBuff[trigP] <=
(dataMax + dataMin) / 2)) {
        break;
      }                                              // negative trigger poshition found !
    }
  }
  trigSync = true;
  if (trigP >= ((REC_LENG / 2) + 50)) {             // If the trigger is not found in
range
    trigP = (REC_LENG / 2);                          // Set it to the center for the time
being
    trigSync = false;                                // set Unsync display flag
  }
  if ((dataMax - dataMin) <= MIN_TRIG_SWING) {      // amplitude of the waveform smaller
than the specified value
    trigSync = false;                                // set Unsync display flag
  }
  freqDuty();
}

void freqDuty() {                                  // detect frequency and duty cycle
value from waveform data
  int swingCenter;                                 // center of wave (half of p-p)
  float p0 = 0;                                    // 1-st posi edge
  float p1 = 0;                                    // total length of cycles
  float p2 = 0;                                    // total length of pulse high time
  float pFine = 0;                                 // fine position (0-1.0)
  float lastPosiEdge;                              // last positive edge position

  float pPeriod;                                   // pulse period
  float pWidth;                                    // pulse width

  int p1Count = 0;                                 // wave cycle count
  int p2Count = 0;                                 // High time count

  boolean a0Detected = false;
  //  boolean b0Detected = false;
  boolean posiSerch = true;                        // true when serching posi edge

  swingCenter = (3 * (dataMin + dataMax)) / 2;  // calculate wave center value

  for (int i = 1; i < REC_LENG - 2; i++) {      // scan all over the buffer
    if (posiSerch == true) {   // posi slope (frequency serch)
      if ((sum3(i) <= swingCenter) && (sum3(i + 1) > swingCenter)) {  // if across the
center when rising (+-3data used to eliminate noize)
        pFine = (float)(swingCenter - sum3(i)) / ((swingCenter - sum3(i)) + (sum3(i +
1) - swingCenter) );  // fine cross point calc.
        if (a0Detected == false) {               // if 1-st cross
          a0Detected = true;                     // set find flag
          p0 = i + pFine;                        // save this position as startposition
        } else {
```

```
          p1 = i + pFine - p0;                  // record length (length of n*cycle
time)
          p1Count++;
        }
        lastPosiEdge = i + pFine;               // record location for Pw calcration
        posiSerch = false;
      }
    } else {   // nega slope serch (duration serch)
      if ((sum3(i) >= swingCenter) && (sum3(i + 1) < swingCenter)) {  // if across the
center when falling (+-3data used to eliminate noize)
        pFine = (float)(sum3(i) - swingCenter) / ((sum3(i) - swingCenter) +
(swingCenter - sum3(i + 1)) );
        if (aODetected == true) {
          p2 = p2 + (i + pFine - lastPosiEdge); // calucurate pulse width and
accumurate it
          p2Count++;
        }
        posiSerch = true;
      }
    }
  }

  pPeriod = p1 / p1Count;                 // pulse period
  pWidth  = p2 / p2Count;                 // palse width

  waveFreq = 1.0 / ((pgm_read_float(hRangeValue + hRange) * pPeriod) / 25.0); //
frequency
  waveDuty = 100.0 * pWidth / pPeriod;                                        // duty
ratio
}

int sum3(int k) {       // Sum of before and after and own value
  int m = waveBuff[k - 1] + waveBuff[k] + waveBuff[k + 1];
  return m;
}

void startScreen() {                    // Start up screen
  oled.clearDisplay();
//  oled.setTextSize(2);                // at double size character
  oled.setTextColor(WHITE);
  oled.println(F("PMO-RP1 v3.10 on R909"));   // Title(Poor Man's Osilloscope,
RadioPench 1)
  oled.println(F("radiopench/kpa"));          // this for SH1106
  oled.display();                       // actual display here
  delay(1500);
  oled.clearDisplay();
  oled.setTextSize(1);                  // After this, standard font size
}

void dispHold() {                       // display "Hold"
  oled.fillRect(42, 11, 24, 8, BLACK);  // black paint 4 characters
  oled.setCursor(42, 11);
  oled.print(F("Hold"));                // Hold
  oled.display();                       //
}

void dispInf() {                        // Display of various information
  float volt;
  // display DC/AC couple mode
  oled.setCursor(0, 0);
  if (inMode == 0) {
    oled.print(F("DC"));
  } else {
    oled.print(F("AC"));
```

```
}

// vertical sensitivity
oled.setCursor(15, 0);                  // around top left
oled.print(vScale);                     // vertical sensitivity value
if (scopeP == 0) {                      // if scoped
  oled.drawFastHLine(13, 7, 27, WHITE); // display scoped mark at the bottom
  oled.drawFastVLine(13, 5,  2, WHITE);
  oled.drawFastVLine(39, 5,  2, WHITE);
}

// horizontal sweep speed
oled.setCursor(42, 0);                  //
oled.print(hScale);                     // display sweep speed (time/div)
if (scopeP == 1) {                      // if scoped
  oled.drawFastHLine(40, 7, 33, WHITE); // display scoped mark
  oled.drawFastVLine(40, 5,  2, WHITE);
  oled.drawFastVLine(72, 5,  2, WHITE);
}

// trigger polarity
oled.setCursor(75, 0);                  // at top center
if (trigD == 0) {                       // if positive
  oled.print(char(0x18));               // up mark
} else {
  oled.print(char(0x19));               // down mark
}
if (scopeP == 2) {                      // if scoped
  oled.drawFastHLine(72, 7, 11, WHITE); // display scoped mark
  oled.drawFastVLine(72, 5,  2, WHITE);
  oled.drawFastVLine(82, 5,  2, WHITE);
}

// average voltage
if (inMode == 0) {                      // if DC mode
  oled.setCursor(86, 0);
  oled.print(F("av"));                  // av : average
  if (att10x == 1) {                    // if 10x attenuator is used
    volt = dataAve * lsb50V / 10.0;     //  range value
  } else {                              // no!
    volt = dataAve * lsb5V / 10.0;      // 5V range value
  }
  if (volt < 10.0) {                    // if less than 10V
    dtostrf(volt, 4, 2, chrBuff);       // format x.xx
  } else {                              // no! over 10
    dtostrf(volt, 4, 1, chrBuff);       // format xx.x
  }
} else {                                // AC mode
  oled.setCursor(86, 0);
  oled.print(F("rm"));                  // rm : rms root mean square

  if (att10x == 1) {                    // if 10x attenuator is used
    volt = dataRms * lsb50Vac;          //      l 50V range value
  } else {                              // no!
    volt = dataRms * lsb5Vac;           // 5V range value
  }

  if (volt < 10.0) {                    // if less than 10V
    dtostrf(volt, 4, 2, chrBuff);       // format x.xx
  } else {                              // no!
    dtostrf(volt, 4, 1, chrBuff);       // format xx.x
  }
}
oled.setCursor(98, 0);                  // at top right
```

```
oled.print(chrBuff);                    //  d  l  ¥   display voltage
oled.print(F("V"));

// vartical scale lines
volt = rangeMaxDisp / 100.0;            // convert Max voltage
if (vRange <= 3) {                      // 2if range is 20 or more
  dtostrf(volt, 4, 0, chrBuff);         // format ** @
} else {
  if (vRange <= 7) {
    dtostrf(volt, 4, 1, chrBuff);       // format **.* @
  } else {
    dtostrf(volt, 4, 2, chrBuff);       // format *.** @
  }
}
oled.setCursor(0, 9);
oled.print(chrBuff);                    // display Max value

volt = (rangeMaxDisp + rangeMinDisp) / 200.0; // center value calculation
if (vRange <= 3) {                      // 20V
  dtostrf(volt, 4, 0, chrBuff);         // format **
} else {
  if (vRange <= 7) {
    dtostrf(volt, 4, 1, chrBuff);       // format **
  } else {
    dtostrf(volt, 4, 2, chrBuff);       // format *.*
  }
}
oled.setCursor(0, 33);
oled.print(chrBuff);                    // display the value

volt = rangeMinDisp / 100.0;            // convart Min voltage
if (vRange <= 3) {                      // 20V
  dtostrf(volt, 4, 0, chrBuff);         // format **
} else {
  if (vRange <= 7) {
    dtostrf(volt, 4, 1, chrBuff);       // format **.
  } else {
    dtostrf(volt, 4, 2, chrBuff);       // format *.*
    if (inMode >= 1 ) {                 // compress zero(-0.25 -> -.25)
      chrBuff[1] = chrBuff[2];          //
      chrBuff[2] = chrBuff[3];
      chrBuff[3] = chrBuff[4];
      chrBuff[4] = chrBuff[5];
    }
  }
}
oled.setCursor(0, 57);
oled.print(chrBuff);                    // display the value

// display frequency, duty % or trigger missed
if (trigSync == false) {                // If trigger point can't found
  oled.fillRect(92, 14, 24, 8, BLACK);  // black paint 4 character
  oled.setCursor(92, 14);               //
  oled.print(F("unSync"));              // dosplay Unsync
} else {
  oled.fillRect(91, 12, 25, 9, BLACK);  // erase Freq area
  oled.setCursor(92, 13);               // set display locatio
  if (waveFreq < 99.9) {                // if less than 99.9Hz
    oled.print(waveFreq, 1);            // display 99.9Hz
    oled.print(F("Hz"));
  } else if (waveFreq < 999.9) {        // if less than 999.9Hz
    oled.print(waveFreq, 0);            // display 999Hz
    oled.print(F("Hz"));
  } else if (waveFreq < 9999.9) {       // if less than 9.9999kHz
```

```
      oled.print((waveFreq / 1000.0), 2); // display 9.99kH
      oled.print(F("kH"));
    } else {                              // if more
      oled.print((waveFreq / 1000.0), 1); // display 99.9kH
      oled.print(F("kH"));
    }
    oled.fillRect(97, 21, 25, 10, BLACK); // erase Freq area (as small as possible)
    oled.setCursor(98, 23);               // set location
    oled.print(waveDuty, 1);              // display duty (High level ratio) in %
    oled.print(F("%"));
  }
  // this for debug (value display on screen)
//    oled.fillRect(40, 12, 25, 9, BLACK);    //
//    oled.setCursor(40, 13);                 //
//    oled.print(re_result);                  //
}

void plotData() {                     // plot waveform on OLED
  long y1, y2;
  if (hRange <= 9) {                  //noromal plot
    for (int x = 0; x <= 98; x++) {
      y1 = map(waveBuff[x + trigP - 50], rangeMin, rangeMax, 63, 9); // convert to plot
address
      y1 = constrain(y1, 9, 63);                                     // Crush(Saturate)
the protruding part
      y2 = map(waveBuff[x + trigP - 49], rangeMin, rangeMax, 63, 9); // to address
calucurate
      y2 = constrain(y2, 9, 63);                                     //
      oled.drawLine(x + 27, y1, x + 28, y2, WHITE);                  // connect between
point
    }
  } else if (hRange == 10) {          // 100us     W Â 2 { g  ¥   zoom 2X when 100us
range
    for (int x = 0; x <= 49; x++) {
      y1 = map(waveBuff[x + trigP - 25], rangeMin, rangeMax, 63, 9); // convert to plot
address
      y1 = constrain(y1, 9, 63);                                     // Crush(Saturate)
the protruding part
      y2 = map(waveBuff[x + trigP - 24], rangeMin, rangeMax, 63, 9); // to address
calucurate
      y2 = constrain(y2, 9, 63);                                     //
      oled.drawLine(x * 2 + 27, y1, x * 2 + 29, y2, WHITE);              // connect
between point
    }
  } else if (hRange == 11) {          // 50us     W Â 4 { g  ¥   zoom 4x when 50us range
    for (int x = 0; x <= 24; x++) {
      y1 = map(waveBuff[x + trigP - 13], rangeMin, rangeMax, 63, 9); // convert to plot
address
      y1 = constrain(y1, 9, 63);                                     // Crush(Saturate)
the protruding part
      y2 = map(waveBuff[x + trigP - 12], rangeMin, rangeMax, 63, 9); // to address
calucurate
      y2 = constrain(y2, 9, 63);                                     //
      oled.drawLine(x * 4 + 27, y1, x * 4 + 31, y2, WHITE);              // connect
between point
    }
  }
}

void saveEEPROM() {                   // Save the setting value in EEPROM after
waiting a while after the button operation.
  if (saveTimer > 0) {                // If the timer value is positive,
    saveTimer = saveTimer - timeExec; // Timer subtraction
    if (saveTimer < 0) {              // if time up
```

```
      EEPROM.write(0, vRange);          // save current status to EEPROM
      EEPROM.write(1, hRange);
      EEPROM.write(2, trigD);
      EEPROM.write(3, scopeP);
      EEPROM.write(4, offset5Vac >> 8);    //
      EEPROM.write(5, offset5Vac & 0xFF);  //
      EEPROM.write(6, offset50Vac >> 8);   //      50V
      EEPROM.write(7, offset50Vac & 0xFF); //
    }
  }
}

void loadEEPROM() {                     // Read setting values from EEPROM (abnormal
values will be corrected to default)
  int x;
  x = EEPROM.read(0);                   // vRange
  if ((x < 0) || (x > 9)) {             // if out side 0-9
    x = 3;                              // default value
  }
  vRange = x;

  x = EEPROM.read(1);                   // hRange
  if ((x < 0) || (x > 11)) {            // if out of 0-11
    x = 3;                              // default value
  }
  hRange = x;
  x = EEPROM.read(2);                   // trigD
  if ((x < 0) || (x > 1)) {             // if out of 0-1
    x = 1;                              // default value
  }
  trigD = x;
  x = EEPROM.read(3);                   // scopeP
  if ((x < 0) || (x > 2)) {             // if out of 0-2
    x = 1;                              // default value
  }
  scopeP = x;

  x = EEPROM.read(4);                   // offset value of AC5V
  x = x << 8 ;
  x = x | EEPROM.read(5);
  if ((x < 350) || (x > 650 )) {        // if abnormal value,
    x = 594;                            // default value
  }
  offset5Vac = x;

  x = EEPROM.read(6);                   // offset value of AC50V
  x = x << 8 ;
  x = x | EEPROM.read(7);
  if ((x < 350) || (x > 650 )) {        // if abnormal value,
    x = 546;                            // default value
  }
  offset50Vac = x;
}

void auxFunctions() {                   // select AUX function
  if (digitalRead(RE_SW) == LOW) {      // if SELECT button pushed, measure battery
voltage
    battVolt();
  }
  if (analogRead(FUNC_SW) <400) {       //
    dmm5V();
  }
  if (analogRead(FUNC_SW) <800) {        //
    dmm50V();
```

```
  }
}

void battVolt() {                          // Battery voltage measure (this for pen
osillo)
  float volt;
  long x;
  analogReference(DEFAULT);                // ADC full scale set to Vcc
  while (1) {                              // do forever
    x = 0;
    for (int i = 0; i < 100; i++) {        // 100 times
      x = x + analogRead(1);               // A11 pin voltage and accumulate
    }
    volt = (x / 100.0) * 5.0 / 1023.0;     // convert voltage value
    oled.clearDisplay();                   // all erase screen(0.4ms)
    oled.setTextColor(WHITE);              // write in white character
    oled.setCursor(20, 16);                //
    oled.setTextSize(1);                   // standerd size character
    oled.println(F("Battery voltage"));
    oled.setCursor(35, 30);                //
    oled.setTextSize(2);                   // double size character
    dtostrf(volt, 4, 2, chrBuff);          // display batterry voltage x.xxV
    oled.print(chrBuff);
    oled.println(F("V"));
    oled.display();
    delay(150);
  }
}

void dmm5V() {                             // digital voltmeter 5V range
  float volt, vPP;
  analogReference(INTERNAL);
  while (1) {                              //
    digitalWrite(13, HIGH);                // flash LED
    oled.clearDisplay();                   // erase screen (0.4ms)
    oled.setTextColor(WHITE);              // write in white character
    oled.setCursor(0, 0);                  //
    oled.setTextSize(1);                   // by standerd size character

    if (digitalRead(7) == HIGH) {          // if awitch is DC mode
      hiZ(R_820k);                         // set measure condition
      hiZ(R_82k);
      hiZ(R_12k);                          //
      volt = analogRead(0) * lsb5V;        // measure voltage
      oled.println(F("DC DVM 5V range"));  //
      oled.setCursor(20, 22);              //
      oled.setTextSize(2);                 // double size character
      dtostrf(volt, 4, 2, chrBuff);        // display voltage x.xxV
      oled.print(chrBuff);
      oled.print(F("V"));
    } else {                               // AC mode
      pull5V(R_820k);                      // give offset
      hiZ(R_82k);
      hiZ(R_12k);                          // Set the attenuator control pin to Hi-z
(use as input)

      ADCSRA = ADCSRA & 0xf8;              // clear bottom 3bit
      ADCSRA = ADCSRA | 0x07;              // dividing ratio = 128 (default of Arduino
j
      for (int i = 0; i < REC_LENG; i++) { // recoord to buffer at 5ms range settings
        waveBuff[i] = analogRead(0);       // read and save approx 112  s
        delayMicroseconds(87);             // timing adjustmet tuned
      }
      dataAnalize();                       // analize data
```

```
      volt = dataRms * lsb5Vac;             // get RMS value
      vPP = (dataMax - dataMin) * lsb5Vac; // Peak to peak voltage

      oled.println(F("AC DVM 5V range"));
      oled.setTextSize(2);                  // double size character
      dtostrf(volt, 4, 2, chrBuff);         // foromat x.xx
      oled.setCursor(20, 16);               //
      oled.print(chrBuff);                  // display rms voltage
      oled.println(F("Vrms"));
      dtostrf(vPP, 4, 2, chrBuff);          // format x.xx
      oled.setCursor(20, 38);               //
      oled.print(chrBuff);                  //
      oled.println(F("Vpp"));
    }
    oled.display();                         // actual display here
    digitalWrite(13, LOW);                  // stop LED flash
    delay(150);                             // wait next measure
  }
}

void dmm50V() {                             // digital voltmeter 5V range
  float volt, vPP;
  analogReference(INTERNAL);
  while (1) {                               // forever,
    digitalWrite(13, HIGH);                 // flash LED
    oled.clearDisplay();                    // erase screen (0.4ms)
    oled.setTextColor(WHITE);               // write in white character
    oled.setCursor(0, 0);                   //
    oled.setTextSize(1);                    // by standerd size character

    if (digitalRead(7) == HIGH) {           // if awitch is DC mode
      hiZ(R_820k);                          // set measure condition
      hiZ(R_82k);
      pullGND(R_12k);                       //
      volt = analogRead(0) * lsb50V;        // measure voltage
      oled.println(F("DC DVM 50V range")); //
      oled.setCursor(20, 22);               //
      oled.setTextSize(2);                  // double size character
      dtostrf(volt, 4, 1, chrBuff);         // display voltage xx.xV
      oled.print(chrBuff);
      oled.print(F("V"));
    } else {                                // AC mode
      hiZ(R_820k);                          // set measure condition
      pull5V(R_82k);                        // pull up
      pullGND(R_12k);                       // att 10x

      ADCSRA = ADCSRA & 0xf8;               // clear bottom 3bit
      ADCSRA = ADCSRA | 0x07;               // dividing ratio = 128 (default of Arduino
j
      for (int i = 0; i < REC_LENG; i++) { // record to buffer at 5ms range settings
        waveBuff[i] = analogRead(INPUT_PORT);       // read and save approx 112  s
        delayMicroseconds(87);              // timing adjustmet tuned
      }
      dataAnalize();                        // analize data

      volt = dataRms * lsb50Vac;             // get RMS value
      vPP = (dataMax - dataMin) * lsb50Vac; // get Peak to peak voltage

      oled.println(F("AC DVM 50V range"));
      oled.setTextSize(2);                   // double size character
      dtostrf(volt, 4, 1, chrBuff);          // foromat xx.x
      oled.setCursor(20, 16);                //
      oled.print(chrBuff);                   // display rms voltage
```

```
      oled.println(F("Vrms"));
      dtostrf(vPP, 4, 1, chrBuff);          // format xx.x
      oled.setCursor(20, 38);               //
      oled.print(chrBuff);                  // P-P d    ¥
      oled.println(F("Vpp"));
    }
    oled.display();                         // actual display here
    digitalWrite(13, LOW);                  // stop LED flash
    delay(150);                             // wait next measure
  }
}

void uuPinOutputLow(unsigned int d, unsigned int a) { // output LOW at specfied pin
  // PORTx =0, DDRx=1
  // example uuPinOutputLow(0b00000001111000, 0b000000); D3-6 output Low
  unsigned int x;
  x = d & 0x00FF; PORTD &= ~x; DDRD |= x;
  x = d >> 8;     PORTB &= ~x; DDRB |= x;
  x = a & 0x003F; PORTC &= ~x; DDRC |= x;
}

void key_sense() {                  //

  if (digitalRead(RE_SW) == 0 ) {   // if select buton was pushed
    saveTimer = 5000;               // set EEPROM save timer to 5 secnd
    switchPushed = true;            // switch pushed falag ON
  }
  if (digitalRead(RE_SW) == 0) {    // if select button pushed,
    scopeP++;                       // forward scope position
    if (scopeP > 2) {               // if upper limit
      scopeP = 0;                   // move to start position
    }
    delay(100);
  }

  if ( re_result == 0x20 ) {        // if RE was rotated counter clockwise, and
    re_result = 0;
    if (scopeP == 0) {              // scoped vertical range
      vRange++;                     // V-range up !
      if (vRange > 9) {             // if upper limit
        vRange = 9;                 // stay as is
      }
    }
    if (scopeP == 1) {              // if scoped hrizontal range
      hRange++;                     // H-range up !
      if (hRange > 11) {            // if upper limit
        hRange = 11;                // stay as is
      }
    }
    if (scopeP == 2) {              // if scoped trigger porality
      trigD = 0;                    // set trigger porality to +
    }
  }

  if ( re_result == 0x10 ) {        // if RE was rotated clockwise, and
    re_result = 0;
    if (scopeP == 0) {              // scoped vertical range
      vRange--;                     // V-range DOWN
      if (vRange < 2) {             // if bottom (Auto5V Auto50V)
        vRange = 2;                 // stay as is
      }
    }
    if (scopeP == 1) {              // if scoped hrizontal range
      hRange--;                     // H-range DOWN
```

```
      if (hRange < 0) {            // if bottom
        hRange = 0;                // satay as is
      }
    }
    if (scopeP == 2) {             // if scoped trigger porality
      trigD = 1;                   // set trigger porality to -
    }
  }

//  if ((x & 0x08) == 0) {           // if HOLD button(pin11) pushed
//    hold = ! hold;                 // revers the flag
//  }
}
```