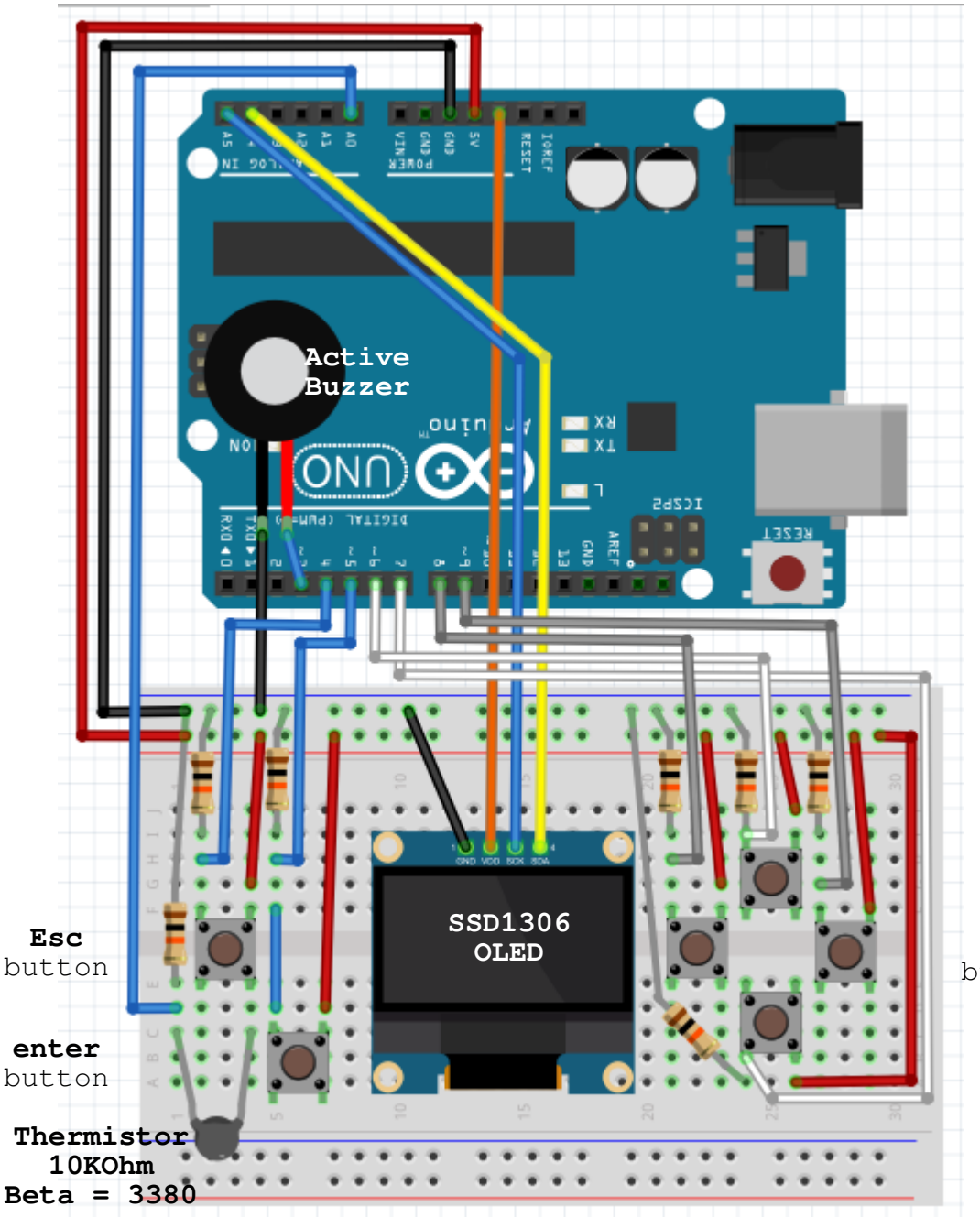
ATMEGA328/ARDUINO - PROJECT - DIGITAL ALARM CLOCK - OLED
https://github.com/teaksoon/p_daco



HARDWARE
1x Computer with Arduino IDE Software
1x USB 2.0 Type A/B Data Cable
1x Arduino Uno Board
1x Solderless Breadboard
Nx Jumper wires

1x Active Buzzer
6x Tactile Switch with 6x 10KOhm Resistor
1x SSD1306 OLED Module i2c 64x128 pixel

- Modular Design Extension -

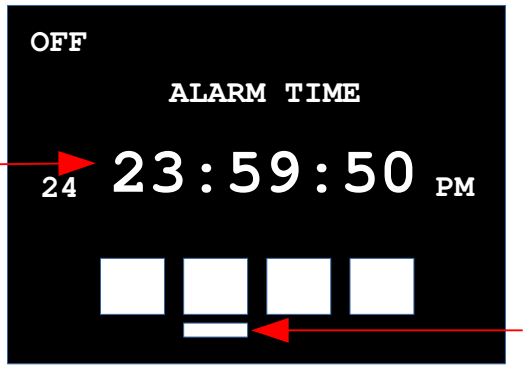
1x 10Kohm Thermistor(Beta=3380) with 1x 10KOhm Resistor

ATMEGA328/ARDUINO – PROJECT – DIGITAL ALARM CLOCK – OLED
https://github.com/teaksoon/p_daco

Source code: **p_daco_clock_alarm**
Download from:
https://github.com/teaksoon/p_daco/blob/main/2022_01_09_p_daco_source.zip
Upload PROGRAM, watch the OLED Screen

While in the MENU MODE and the Navigation Bar is at the CLOCK FUNCTION, the OLED screen will display CLOCK FUNCTION "Live" Clock (updated every 1 second)

Alarm Time



ESC Button – Move Navigation Bar to first MENU OPTION

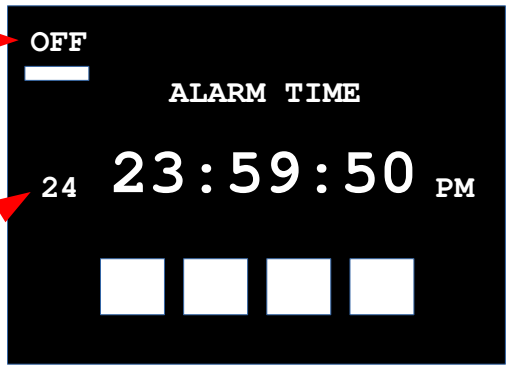
LEFT/RIGHT Button – Move MENU Navigation Bar. Show different FUNCTION Live Data

ENTER Button – Move into FUNCTION EDIT MODE

Presssing ENTER Button while in MENU MODE, will enter the EDIT MODE (ALARM TIME SETTING and alarm ON/OFF Switch)

Alarm Switch ON/OFF Switch must be turned ON for the Alarm trigger

12/24hrs Mode follows the Main Clock



ESC Button – Cancel Edited Data and return to MENU MODE, continue with the previous Data

LEFT/RIGHT Button – Move Navigation Bar

UP/DOWN Button – Change data at the Navigation Bar position

ENTER Button – Save Edited Data and return to MENU MODE, continue from Saved Data

We need turn the Alarm Switch to ON or OFF and set Alarm Trigger time, HH:MM:SS. Alarm is checked every seconds. The code for Alarm Trigger is arranged in sequence below to reduce overhead in the Super Loop processing

```
void alarm_monitor() {
  if (ala_switch == ON) { // check switch, if OFF, quit
    if (ala_ss == clo_ss) { // compare ss, if not match, quit
      if (ala_mm == clo_mm) { // compare mm, if not match, quit
        if (ala_hh == clo_hh) { // compare hh, if not match, quit
          if (ala_ampm == clo_ampm) { // compare am/pm, if not match, quit
            if (buzzer_state == OFF) { // if buzzer is OFF, Trigger Alarm
              // Trigger Alarm
              buzzer_millis_start = millis();
              buzzer_on();
            }
          }
        }
      }
    }
  }
}
```

After the Alarm Buzzer has been triggered, pressing any buttons will stop the Alarm Buzzer

HOUR CHANGE:
In ALARM MODE, it is very similar to the CLOCK Hour Change Mode.
NOTE: Alarm Time uses the clock hhmode (for easier alarm synchro)

We will need to do some adjustments to the Hour Value and the AM/PM state whhen adding or reducing on existing Hour Data

In 24Hours Mode:
Adding 1 Hour to 23 Hours will turn 0 Hours (PM to AM)
Reduce 1 Hour from 0 Hours will turn to 23 Hours (AM to PM)
Anything more than 11 Hour is PM otherwise AM

In 12Hours Mode:
Adding 1 Hour to 11PM will turn 12AM (PM to AM)
Adding 1 Hour to 12PM will turn 1PM (there is no 0 PM, 12 becomes 1)
Adding 1 Hour to 11AM will turn to 12PM (AM to PM)
Reduce 1 Hour from 12AM will turn 11PM (AM to PM)
Reduce 1 Hour from 1PM will turn 12PM (there is no 0 PM, 1 becomes 12)
Reduce 1 Hour from 12PM will turn 11AM (PM t AM)

```
void alarm_edit_hh(int8_t editDir) {
  // Effects hh and ampm
  // Alarm Time uses the clock hhmode ( for easier alarm synchro )
  //
  if (clo_hhmode == HHMODE_24)  {
    // Currently in 24hrs Mode, Cycle between 0 to 23
    ala_hh_t   = g_next_nn(editDir,ala_hh_t,0,23);
    ala_ampm_t = (ala_hh_t > 11) ? PM:AM;          // more than 11 = PM
  } else {
    // Currently in 12hrs Mode
    if (editDir == 1) {
        // Increase One Hour in 12hrs Mode
        if (ala_ampm_t == PM) {
          // This is PM
          if (ala_hh_t == 11) ala_ampm_t = AM;     // 11PM to 12AM
          ala_hh_t++;
          if (ala_hh_t > 12)  ala_hh_t = 1;        // After 12 = 1
        } else {
          // This is AM
          if (ala_hh_t == 11) ala_ampm_t = PM;     // 11AM to 12PM
          ala_hh_t++;
          if (ala_hh_t > 12)  ala_hh_t = 1;        // After 12 = 1
        }
    } else {
      if (editDir == -1) {
        // Reduce One Hour in 12hrs Mode
        if (ala_ampm_t == PM) {
          // This is PM
          if (ala_hh_t == 12) ala_ampm_t = AM;     // 12PM to 11AM
          ala_hh_t--;
          if (ala_hh_t < 1)   ala_hh_t = 12;       // Before 1 = 12
        } else {
          // This is AM
          if (ala_hh_t == 12) ala_ampm_t = PM;     // 12AM to 11PM
          ala_hh_t--;
          if (ala_hh_t < 1)   ala_hh_t = 12;       // Before 1 = 12
        }
      }
    }
  }
  sf_show_ampm(ala_ampm_t); // ampm
  g_int_padl(ala_hh_t,tmp_s,2,'0');p_ssd1306_string_L(3,24,tmp_s); // hh
}
```

Changes in the CLOCK 12/24 HOURS MODE will effect the Hour in the ALARM TIME

24Hours Mode to 12Hours Mode:
Anything more than 12Hours needs to be adjusted, minus 12
0 Hours needs to be adjusted to 12

12Hours Mode to 24Hours Mode:
When in PM mode, anything less than 12 need to be adjusted, plus 12
When in AM mode, 12AM needs to be adjusted to 0

```
void clock_edit_hhmode() {
  // Effects hhmode and hh in clock time, hh in alarm time
  //
  if (clo_hhmode_t == HHMODE_24) {
    // Currently 24hrs Mode, changing to 12hrs Mode
    if (clo_hh_t > 12) {
      clo_hh_t = clo_hh_t-12;                    // when more than 12, -12
    } else {
      if (clo_hh_t == 0) clo_hh_t = 12;          // 0 in 24hrs = 12
    }
    // alarm follows clock hhmode. alarm hh change, 24hrs to 12hrs
    if (ala_hh > 12) {
      ala_hh = ala_hh-12;                        // when more than 12, -12
    } else {
      if (ala_hh == 0) ala_hh = 12;              // 0 in 24hrs = 12
    }
    clo_hhmode_t = HHMODE_12;
  } else {
    // Currently 12hrs Mode, changing to 24hrs Mode
    if ( clo_ampm_t == PM) {
      if (clo_hh_t < 12) clo_hh_t = clo_hh_t+12;  // PM, less than 12, +12
    } else {
      if (clo_hh_t == 12) clo_hh_t = 0;           // 12 in 12hrs = 0
    }
    // alarm follows clock hhmode. alarm hh change, 12hrs to 24hrs
    if ( ala_ampm == PM) {
      if (ala_hh < 12)    ala_hh  = ala_hh+12;    // PM, less than 12, +12
    } else {
      if (ala_hh == 12)   ala_hh  = 0;            // 12 in 12hrs = 0
    }
    clo_hhmode_t = HHMODE_24;
  }
  sf_show_hhmode(clo_hhmode_t);  // hhmode
  g_int_padl(clo_hh_t,tmp_s,2,'0');p_ssd1306_string_L(3,24,tmp_s); // hh
}
```

ATMEGA328/ARDUINO - PROJECT - DIGITAL ALARM CLOCK - OLED
https://github.com/teaksoon/p_daco

At this stage,our OLED Digital Clock is fully funtional with Alarm feature

Clock - Live Running Clock showing Time in 12/24Hours Mode, Date
(Automatically updated every seconds with auto AM/PM adjustments)

Clock - Time/Date Editing with 12/24Hours Mode Change with auto Hour, AM/PM adjustments

Alarm Clock - ON/OFF Switch and Alarm Time Display/Editing

Alarm Monitoring - Trigger Alarm when Alarm Time and Clock Time matches

Utility - To set and test Alarm Buzzer duration time in seconds

Below is the amount of PROGRAM MEMORY and SRAM MEMORY from the ATMEGA328 micro-controller that has been used by our PROGRAM (PROGRAM codes that are not optimized yet, means we can still reduce)

```
Done uploading.

Sketch uses 7076 bytes (21%) of program storage space. Maximum is 32256 bytes.
Global variables use 1421 bytes (69%) of dynamic memory, leaving 627 bytes for local variables. Maximum is 2048 bytes.
```

We still have plenty of PROGRAM STORAGE MEMORY space left, since we only used 21%. So, no worries about our PROGRAM MEMORY. However, the SRAM could be an issue because we have already used 1421 bytes out of our total 2048 btyes (69%) of our SRAM Memory

We need to keep track of the SRAM Memory usage from now because **we still have a Thermometer to show.** To get temperature from our Thermistor, will involve some math/calculation, that will surely increase our SRAM usage

After the Thermometer. If we still have enough SRAM, we can put in a Live Analog Clock display (those Clock Display with a Circle and rotating Hour Minutes and Seconds arms inside)

NOTE:
1024 bytes out of our total 2048 bytes of our SRAM memory is used for the SSD1306 i2c OLED Screen buffer memory. That is why our SRAM memory usage is so high

In the worse case scenario if we run out of SRAM Memory, we can use the non-buffered SSD1306 i2c OLED Screen coding technique, which will give us back 1024 bytes of SRAM. However, our clock design has to change

The non-buffered SSD1306 i2c OLED Screen technique cannot draw circle or diagonal lines because the SSD1306 OLED i2c Graphics Memory is the “write-only” memory type. We need to have the “read-write” memory type in order to draw circle or diagonal lines