

esp32 oszilloskop Uhr

Datum: 2024-12-04

Oszilloskop im XY-Modus als Uhrenanzeige.

Die Spannungen werden mit dem esp32 erzeugt, weil

- der über WLAN die Uhrzeit holen kann und
- zwei Digital zu Analog Wandler hat.

ESP32 Osziclock using DAC1 and DAC2, pin 25 and 26.

Contents

1 Inbetriebnahme	1
2 TODO	1
2.1 plot different things	2
3 Übernehmen des alten Projekts und testen	2
3.1 Change 01	6
3.2 Change 02	6
3.3 Change 03	7
3.4 Change 04: Sommer/Winterzeit aus NTP	8
3.5 Change 05: Konfiguration	8
3.6 Change 06: xmas mode	8
4 Aktueller Code	10
4.1 esp-oszidisplay.ino	10
4.2 wifitime.h	14
4.3 wlan.h.in	17

1 Inbetriebnahme

Die Datei wlan.h.in in wlan.h kopieren

- SSID und Passwort anpassen.
- ntpServer anpassen, wenn über DHCP ein ntpServer konfiguriert ist, wird der vom DHCP verwendet.
- time_zone anpassen.

2 TODO

- HW Koax-Kabel, Telefonkabel geht auch :-)
- HW Gehäuse ... is only covering the details :-))
- HW Netzteil
- HW 5V aus dem Oszilloskop
- split .h in .h and .cpp
- Konfiguration via serielle Konsole
- Konfiguration in Datei am ESP speichern.

- refetch time once a day
- digital countdown zu Stundenende
- HTL Schriftzug beim Start. Logo ?
- DMA zum DAC mit dac_contious.h ... mehr Punkte ?

2.1 plot different things

get from server/serial/dip-switch

save points to draw by skipping the circle

- christmas tree
- burnings candle/s
- different watch hand: candle, xmastree, ...

3 Übernehmen des alten Projekts und testen

```
// Use an oszilloscope as display

// revision log
// 16-dec-2022

// for brownout detection
#include "soc/soc.h"
#include "soc/rtc_cntl_reg.h"

#include <driver/dac.h>

#include "wifitime.h"

int getLocalTime() {
    struct tm timeinfo;
    int t = -1;
    if (getLocalTime(&timeinfo)) {
        int h = timeinfo.tm_hour;
        if (timeinfo.tm_isdst) {
            h++;
        }
        t = h * 100 + timeinfo.tm_min;
    }
    return t;
}

#include <ArduinoGraphics.h>

class OSZIGraphics : public ArduinoGraphics {
    uint8_t xy[512][2]; // the display ... the dots to display
public:
    int xy_last = 0;

    OSZIGraphics()
        : ArduinoGraphics(256, 256) {}

    void dump() {
        for (int i = 0; i < xy_last; i++) {
            Serial.print(xy[i][0]);
            Serial.print(" ");
        }
    }
};
```

```

        Serial.println(xy[i][1]);
    }
}

void set(int x, int y, uint8_t r, uint8_t g, uint8_t b) {
    xy[xy_last][0] = x;
    xy[xy_last][1] = y;
    xy_last++;
}

// tick must be called in loop or an isr
// returns true if new screen starts.
bool tick() {
    static int xy_index = 0;
    dac_output_voltage(DAC_CHANNEL_1, xy[xy_index][0]);
    dac_output_voltage(DAC_CHANNEL_2, xy[xy_index][1]);
    xy_index++;
    if (xy_index > xy_last) {
        xy_index = 0;
        return true;
    }
    return false;
}

void line(int x0, int y0, int x1, int y1) {
    // do not draw horizontal from higher to lower x
    if ((x0 > x1) && (y0 == y1)) {
        int x = x0;
        x0 = x1;
        x1 = x;
    }
    // do not draw vertical from higher to lower x
    if ((x0 == x1) && (y0 > y1)) {
        int y = y0;
        y0 = y1;
        y1 = y;
    }
    ArduinoGraphics::line(x0, y0, x1, y1);
}
};

```

```
OSZIGraphics oszi = OSZIGraphics();
```

```

class AnalogClock {
public:
    // index after clock face
    int xy_index_eof = 0;

    int mid[2] = { 100, 100 };
    int watch_rad = 50;
    OSZIGraphics* _oszi;
    AnalogClock(OSZIGraphics* disp) {
        _oszi = disp;
    }

    void drawFace() {
        _oszi->stroke(127, 127, 127); // color
        for (int i = 0; i < 12; i++) {
            float _angle = PI * 2 * i / 12.;
            _oszi->line(mid[0] + watch_rad * cos(_angle),
                      mid[1] + watch_rad * sin(_angle),

```

```

        mid[0] + (watch_rad + 7) * cos(_angle),
        mid[1] + (watch_rad + 7) * sin(_angle));
    int _sub = 10;
    for (int j = 0; j < _sub; j++) {
        float _angle = PI * 2 * (i + (j / float(_sub))) / float(_sub);
        _oszi->line(mid[0] + (watch_rad + 7) * cos(_angle),
                    mid[1] + (watch_rad + 7) * sin(_angle),
                    mid[0] + (watch_rad + 7) * cos(_angle),
                    mid[1] + (watch_rad + 7) * sin(_angle));
    }
}
xy_index_eof = _oszi->xy_last;
}

void drawHand(int len, float angle) {
    int cos_ = len * cos(angle);
    int sin_ = len * sin(angle);
    // BUG line is direction dependend
    _oszi->line(mid[0], mid[1], mid[0] + cos_, mid[1] + sin_);
}

void drawHands(int number) {
    // hhmm
    int mm = number % 100;
    float hh = int(number / 100) + float(mm) / 60;
    _oszi->xy_last = xy_index_eof;

    float mm_angle = PI / 2 + PI * 2 / 60. * (60 - mm);
    drawHand(watch_rad, mm_angle);

    float hh_angle = PI / 2 + PI * 2 / 12. * (12 - hh);
    drawHand(watch_rad - 10, hh_angle);
}

void animation() {
    static int angle = 0;
    _oszi->xy_last = xy_index_eof;
    drawHand(30, angle);
    angle -= 1;
}

void dump() {
    _oszi->dump();
}
};

AnalogClock aClock = AnalogClock(&oszi);

void setup() {
    Serial.begin(115200);
    // brownout detection off ... at least for wifi connect, when on PC USB
    WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0); //disable brownout detector

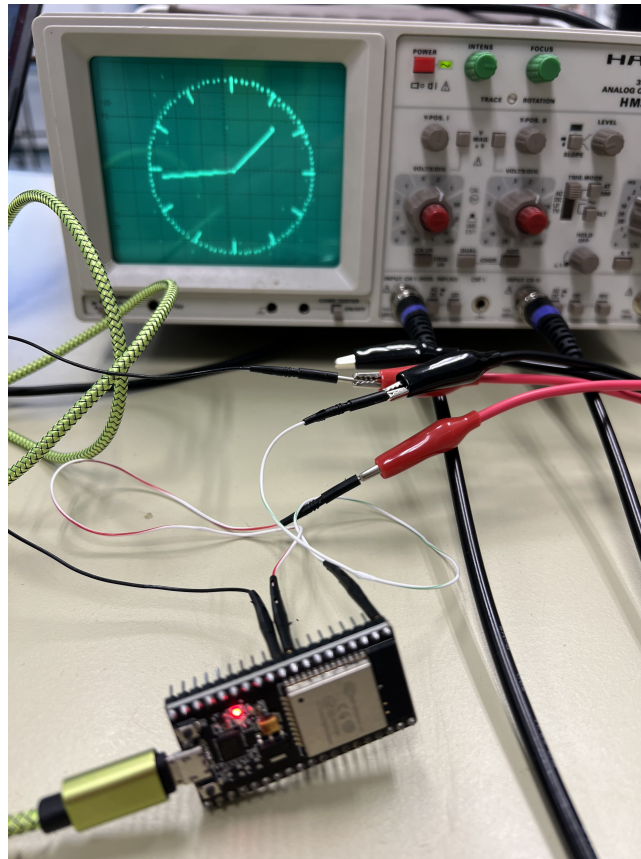
    Serial.println("ESP32 Osziclock using DAC1 and DAC2, pin 25 and 26.");
    if (!oszi.begin()) {
        Serial.println("Failed to initialize the display!");
        // impossible
    }
    Serial.print("Draw clock face:");
    aClock.drawFace();
    Serial.print(aClock.xy_index_eof);
}

```

```
Serial.println("  points");

Serial.println("\nWiFi setup");
wifiSetup();
dac_output_enable(DAC_CHANNEL_1);
dac_output_enable(DAC_CHANNEL_2);
}

void loop() {
  static bool fetch_time = true;
  if (fetch_time) {
    if (getTime_from_network()) {
      fetch_time = false;
    }
  }
  if (oszi.tick()) {
    // one page for a watch is 10ms
    static int cnt = 32000;
    if (cnt > 1000) { // do not fetch time always
      cnt = 0;
      if (fetch_time) {
        aClock.animation();
      } else {
        static int time_ = 0;
        int t = getLocalTime();
        if (t != time_) {
          time_ = t;
          Serial.print("localtime ");
          Serial.println(time_);
          aClock.drawHands(time_);
        }
      }
    }
    cnt++;
  }
}
```



funktioniert aber es gibt Warnungen

3.1 Change 01

Der Code ist unter Versionkontrolle, deshalb entfernen wir den manuellen Zeitstempel und lassen das die Versionkontrolle machen.

```
+++ esp-oszidisplay.ino          (Arbeitskopie)
@@ -1,7 +1,6 @@
// Use an oszilloscope as display

-// revision log
-// 16-dec-2022
+// $Date: 2024-12-04 13:21:56 +0100 (Mi, 04. Dez 2024) $

// for brownout detection
```

3.2 Change 02

Warning: 'DAC_CHANNEL_1' is deprecated: please use 'DAC_CHAN_0' instead

Warning: 'DAC_CHANNEL_2' is deprecated: please use 'DAC_CHAN_1' instead

sind einfach.

Warning: The legacy DAC driver is deprecated, please use *driver/dac_oneshot.h*, *driver/dac_cosine.h* or *driver/dac_continuous.h* instead"

Wechsel zu *dac_oneshot* ... auf <https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/peripherals/dac.html> steht

Direct Voltage Output (One-shot/Direct Mode)

The DAC channels in the group can convert an 8-bit digital value into the analog when *dac_oneshot_output_voltage()* is called (it can be called in ISR). The analog voltage is kept on the DAC channel until the next conversion starts. To start the voltage conversion, the DAC channels need to be enabled first through registering by *dac_oneshot_new_channel()*.

```

--- esp-oszidisplay.ino (Revision 528)
+++ esp-oszidisplay.ino (Arbeitskopie)
@@ -7,10 +7,12 @@
#include "soc/soc.h"
#include "soc/rtc_cntl_reg.h"

-#include <driver/dac.h>
+#include <driver/dac_oneshot.h>

#include "wifitime.h"

+dac_oneshot_handle_t DAC[2];
+
int getLocalTime() {
    struct tm timeinfo;
    int t = -1;
@@ -53,8 +55,8 @@
// returns true if new screen starts.
bool tick() {
    static int xy_index = 0;
-    dac_output_voltage(DAC_CHAN_0, xy[xy_index][0]);
-    dac_output_voltage(DAC_CHAN_1, xy[xy_index][1]);
+    dac_oneshot_output_voltage(DAC[0], xy[xy_index][0]);
+    dac_oneshot_output_voltage(DAC[1], xy[xy_index][1]);
    xy_index++;
    if (xy_index > xy_last) {
        xy_index = 0;
@@ -165,8 +167,11 @@

    Serial.println("\nWiFi setup");
    wifiSetup();
-    dac_output_enable(DAC_CHAN_0);
-    dac_output_enable(DAC_CHAN_1);
+    dac_oneshot_config_t dacfg;
+    dacfg.chan_id = DAC_CHAN_0;
+    dac_oneshot_new_channel(&dacfg, &DAC[0]);
+    dacfg.chan_id = DAC_CHAN_1;
+    dac_oneshot_new_channel(&dacfg, &DAC[1]);
}

```

void loop() {

Compiliert und läuft.

In der Schule startet der esp immer wieder, ... WLANverbindung ?

3.3 Change 03

Warning: sntp.h in IDF's lwip port folder is deprecated. Please include esp_sntp.h

Warning: void sntp_servermode_dhcp(int) is deprecated: use esp_sntp_servermode_dhcp() instead

```

--- wifitime.h (Revision 530)
@@ -3,3 +3,3 @@
#include "time.h"
-#include "sntp.h" // for ntp timezone and daylight savings
+#include "esp_sntp.h" // for ntp timezone and daylight savings

@@ -21,3 +21,3 @@
    configTzTime(time_zone, ntpServer);
-    sntp_servermode_dhcp(1);
+    esp_sntp_servermode_dhcp(true);
    reconnect_count--;

```

Kompiliert und läuft.

Leider immer noch Winterzeit.

3.4 Change 04: Sommer/Winterzeit aus NTP

Die Uhr geht im Sommer eine Stunde vor.

Die Korrektur ausbauen, dann stimmt die Zeit jetzt.

Warten auf Winterzeit.

```
--- esp-oszidisplay.ino          (Revision 537)
@@ -15,3 +15,4 @@

-int getLocalTime() {
+// internal time format 4 digit HHMM
+int getLocalTime_HHMM() {
+    struct tm timeinfo;
@@ -20,5 +21,2 @@
+    int h = timeinfo.tm_hour;
-    if (timeinfo.tm_isdst) {
-        h++;
-    }
+    t = h * 100 + timeinfo.tm_min;
@@ -193,3 +191,3 @@
+    static int time_ = 0;
-    int t = getLocalTime();
+    int t = getLocalTime_HHMM();
+    if (t != time_) {
```

Die Funktion umbenennen ... vielleicht wäre `getTime_in_HHMM` noch klarer.

Tip

Die arduino-IDE hat das rename-refactoring eingebaut, Taste F2. Es gibt auch ein Preview Strg-Enter.

3.5 Change 05: Konfiguration

Wir brauchen einen Platz für die Konfiguration, eine Variable `Config`.

Als erstes bauen wir die Möglichkeit ein Markierungen auf Minuten zu machen.

in C

```
struct {
    bool minute_ticks;
} Config = {
    true
};
```

- Mit `struct` definieren wir einen Datentyp mit verschiedenen benannten Einträgen
- machen gleich eine Variable mit dem Namen `Config` und
- initialisieren mit der Liste `{ true }`.

Wir können jetzt mit `Config.minute_ticks` auf den Wert zugreifen. Davon abhängige Code-Teile fassen wir in

```
if (Config.minute_ticks) {
    // nur wenn true
}
```

ein.

3.6 Change 06: xmas mode

Die Einstellung `bool Config.xmas` hinzufügen.


```

+++ esp-oszidisplay.ino (Arbeitskopie)
@@ -12,8 +12,9 @@

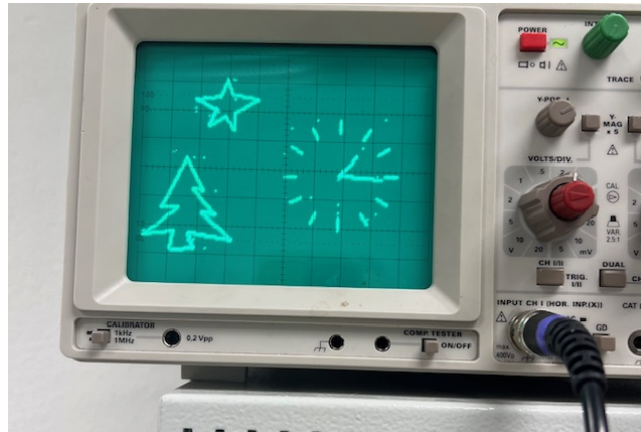
    struct {
        bool minute_ticks;
+   bool xmas;
    } Config = {
-   false
+   false, true
    };

    // digital to analog channel handles
@@ -102,6 +103,47 @@

    void drawFace() {
        _oszi->stroke(127, 127, 127); // color
+   if (Config.xmas) {
+       // draw the tree
+       int lines[] = {
+           // x y
+           30, 10, 32, 20,
+           32, 20, 15, 18,
+           16, 18, 34, 34,
+           34, 34, 25, 34,
+           25, 34, 38, 63, // tip
+           38, 63, 48, 44,
+           48, 44, 42, 45,
+           42, 45, 55, 30,
+           55, 30, 47, 31,
+           47, 31, 64, 15,
+           64, 15, 40, 19,
+           40, 19, 43, 10,
+           43, 10, 31, 9,
+           // star
+           50, 82, 54, 92,
+           55, 90, 42, 97, // tip2
+           42, 97, 56, 99,
+           56, 99, 60, 109, // tip 3
+           60, 109, 63, 100,
+           64, 100, 77, 98, // tip 4
+           77, 98, 64, 90,
+           64, 90, 64, 80, // tip 5
+           64, 80, 59, 88,
+           59, 88, 50, 82 // tip 1
+       };
+       int MAX_LINE = sizeof(lines) / sizeof(lines[0]);
+       for (int i = 0; i < MAX_LINE; i += 4) {
+           _oszi->line(lines[i],
+                       lines[i + 1],
+                       lines[i + 2],
+                       lines[i + 3]);
+       }
+       // shift the watch and scale down
+       mid[0] = 125;
+       mid[1] = 55;
+       watch_rad = 25;
+   }
+   for (int i = 0; i < 12; i++) {
+       float _angle = PI * 2 * i / 12.;
+       _oszi->line(mid[0] + watch_rad * cos(_angle),

```

Wir geben Baum und Stern als Linien mit Anfangs- und Endpunkt ein.



4 Aktueller Code

4.1 esp-oszidisplay.ino

```
// Use an oszilloscope as display

// $Date: 2024-12-04 13:21:56 +0100 (Mi, 04. Dez 2024) $

// for brownout detection
#include "soc/soc.h"
#include "soc/rtc_cntl_reg.h"

#include <driver/dac_oneshot.h>

#include "wifitime.h"

struct {
    bool minute_ticks;
    bool xmas;
} Config = {
    false, true
};

// digital to analog channel handles
dac_oneshot_handle_t DAC[2];

// internal time format 4 digit HHMM
int getLocalTime_HHMM() {
    struct tm timeinfo;
    int t = -1;
    if (getLocalTime(&timeinfo)) {
        int h = timeinfo.tm_hour;
        t = h * 100 + timeinfo.tm_min;
    }
    return t;
}

#include <ArduinoGraphics.h>

class OSZIGraphics : public ArduinoGraphics {
    uint8_t xy[512][2]; // the display ... the dots to display
public:
    int xy_last = 0;
```

```

OSZIGraphics()
: ArduinoGraphics(256, 256) {
}

void dump() {
    for (int i = 0; i < xy_last; i++) {
        Serial.print(xy[i][0]);
        Serial.print(" ");
        Serial.println(xy[i][1]);
    }
}

void set(int x, int y, uint8_t r, uint8_t g, uint8_t b) {
    xy[xy_last][0] = x;
    xy[xy_last][1] = y;
    xy_last++;
}

// tick must be called in loop or an isr
// returns true if new screen starts.
bool tick() {
    static int xy_index = 0;
    dac_oneshot_output_voltage(DAC[0], xy[xy_index][0]);
    dac_oneshot_output_voltage(DAC[1], xy[xy_index][1]);
    xy_index++;
    if (xy_index > xy_last) {
        xy_index = 0;
        return true;
    }
    return false;
}

void line(int x0, int y0, int x1, int y1) {
    // do not draw horizontal from higher to lower x
    if ((x0 > x1) && (y0 == y1)) {
        int x = x0;
        x0 = x1;
        x1 = x;
    }
    // do not draw vertical from higher to lower x
    if ((x0 == x1) && (y0 > y1)) {
        int y = y0;
        y0 = y1;
        y1 = y;
    }
    ArduinoGraphics::line(x0, y0, x1, y1);
}

};

OSZIGraphics oszi = OSZIGraphics();

class AnalogClock {
public:
    // index after clock face
    int xy_index_eof = 0;

    int mid[2] = { 100, 100 };
    int watch_rad = 50;
    OSZIGraphics* _oszi;
    AnalogClock(OSZIGraphics* disp) {
        _oszi = disp;
    }
};

```

```

}

void drawFace() {
  _oszi->stroke(127, 127, 127); // color
  if (Config.xmas) {
    // draw the tree
    int lines[] = {
      // x y
      30, 10, 32, 20,
      32, 20, 15, 18,
      16, 18, 34, 34,
      34, 34, 25, 34,
      25, 34, 38, 63, // tip
      38, 63, 48, 44,
      48, 44, 42, 45,
      42, 45, 55, 30,
      55, 30, 47, 31,
      47, 31, 64, 15,
      64, 15, 40, 19,
      40, 19, 43, 10,
      43, 10, 31, 9,
      // star
      50, 82, 54, 92,
      55, 90, 42, 97, // tip2
      42, 97, 56, 99,
      56, 99, 60, 109, // tip 3
      60, 109, 63, 100,
      64, 100, 77, 98, // tip 4
      77, 98, 64, 90,
      64, 90, 64, 80, // tip 5
      64, 80, 59, 88,
      59, 88, 50, 82 // tip 1
    };
    int MAX_LINE = sizeof(lines) / sizeof(lines[0]);
    for (int i = 0; i < MAX_LINE; i += 4) {
      _oszi->line(lines[i],
        lines[i + 1],
        lines[i + 2],
        lines[i + 3]);
    }
    // shift the watch and scale down
    mid[0] = 125;
    mid[1] = 55;
    watch_rad = 25;
  }
  for (int i = 0; i < 12; i++) {
    float _angle = PI * 2 * i / 12.;
    _oszi->line(mid[0] + watch_rad * cos(_angle),
      mid[1] + watch_rad * sin(_angle),
      mid[0] + (watch_rad + 7) * cos(_angle),
      mid[1] + (watch_rad + 7) * sin(_angle));
    if (Config.minute_ticks) {
      int _sub = 10;
      for (int j = 0; j < _sub; j++) {
        float _angle = PI * 2 * (i + (j / float(_sub))) / float(_sub);
        _oszi->line(mid[0] + (watch_rad + 7) * cos(_angle),
          mid[1] + (watch_rad + 7) * sin(_angle),
          mid[0] + (watch_rad + 7) * cos(_angle),
          mid[1] + (watch_rad + 7) * sin(_angle));
      }
    }
  }
}

```

```

    }
    xy_index_eof = _oszi->xy_last;
}

void drawHand(int len, float angle) {
    int cos_ = len * cos(angle);
    int sin_ = len * sin(angle);
    // BUG line is direction dependend
    _oszi->line(mid[0], mid[1], mid[0] + cos_, mid[1] + sin_);
}

void drawHands(int number) {
    // hmmm
    int mm = number % 100;
    float hh = int(number / 100) + float(mm) / 60;
    _oszi->xy_last = xy_index_eof;

    float mm_angle = PI / 2 + PI * 2 / 60. * (60 - mm);
    drawHand(watch_rad, mm_angle);

    float hh_angle = PI / 2 + PI * 2 / 12. * (12 - hh);
    drawHand(watch_rad - 10, hh_angle);
}

void animation() {
    static int angle = 0;
    _oszi->xy_last = xy_index_eof;
    // snowfall
    drawHand(30, angle);
    angle -= 1;
}

void dump() {
    _oszi->dump();
}
};

AnalogClock aClock = AnalogClock(&oszi);

void setup() {
    Serial.begin(115200);
    // brownout detection off ... at least for wifi connect, when on PC USB
    WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0); //disable brownout detector

    Serial.println("ESP32 Osziclock using DAC1 and DAC2, pin 25 and 26.");
    if (!oszi.begin()) {
        Serial.println("Failed to initialize the display!");
        // impossible
    }
    Serial.print("Draw clock face:");
    aClock.drawFace();
    Serial.print(aClock.xy_index_eof);
    Serial.println("  points");

    Serial.println("\nWiFi setup");
    wifiSetup();
    dac_one_shot_config_t dacfg;
    dacfg.chan_id = DAC_CHAN_0;
    dac_one_shot_new_channel(&dacfg, &DAC[0]);
    dacfg.chan_id = DAC_CHAN_1;
    dac_one_shot_new_channel(&dacfg, &DAC[1]);

```

```

}

void loop() {
    static bool fetch_time = true;
    // TODO refetch_time
    if (fetch_time) {
        if (getTime_from_network()) {
            fetch_time = false;
        }
    }
    if (oszi.tick()) {
        // one page for a watch is 10ms
        static int cnt = 32000;
        if (cnt > 1000) { // do not fetch time always
            cnt = 0;
            if (fetch_time) {
                aClock.animation();
            } else {
                static int time_ = 0;
                int t = getLocalTime_HHMM();
                if (t != time_) {
                    time_ = t;
                    Serial.print("localtime ");
                    Serial.println(time_);
                    aClock.drawHands(time_);
                }
            }
        }
        cnt++;
    }
}

```

4.2 wifitime.h

```

// time from wlan
#ifndef _WIFITIME_H_
#define _WIFITIME_H_

#include <WiFi.h>
#include "time.h"
#include "esp_sntp.h" // for ntp timezone and daylight savings

// Replace with your network credentials
//const char* ssid = "SSID";
//const char* password = "passkey";
#include "wlan.h"

const long  gmtOffset_sec = 0;
const int   daylightOffset_sec = 3600;

static int  reconnect_count = 0;
static int  wifi_state = 0;

bool getTime_from_network() {
    switch (wifi_state) {
        case 0:
            // fallback ntp server from configuration. Might be overwritten by DHCP
            configTzTime(time_zone, ntpServer);
            esp_sntp_servermode_dhcp(true);
            reconnect_count--;
            if (reconnect_count <= 0) {

```

```

        // Connect to Wi-Fi
        Serial.print("Connecting to ");
        Serial.println(ssid);
        WiFi.begin(ssid, password);
        reconnect_count = 10;
        wifi_state++;
    }
    break;
case 1:
    static int cnt = 0;
    if (cnt > 500) {
        cnt = 0;
        reconnect_count--;
        Serial.print(":");
        if (reconnect_count <= 0) {
            wifi_state = 0;
        }
    }
    delay(1);
    cnt++;
    // change of wifi_state to 2 is done in callback
    break;
case 2:
    // Init and get the time
    Serial.println("fetching time");
    struct tm timeinfo;
    if (getLocalTime(&timeinfo)) {
        Serial.println(&timeinfo, "%A, %B %d %Y %H:%M:%S");
    }
    wifi_state++;
    break;
case 3:
    //disconnect WiFi as it's no longer needed
    WiFi.disconnect(true);
    WiFi.mode(WIFI_OFF);
    wifi_state = 0;
    return true;
}
return false;
}

void onWiFiEvent(WiFiEvent_t event)
{
    Serial.printf("[WiFi-event] event: %d\n", event);

    switch (event) {
        case ARDUINO_EVENT_WIFI_READY:
            Serial.println("WiFi interface ready");
            break;
        case ARDUINO_EVENT_WIFI_SCAN_DONE:
            Serial.println("Completed scan for access points");
            break;
        case ARDUINO_EVENT_WIFI_STA_START:
            Serial.println("WiFi client started");
            break;
        case ARDUINO_EVENT_WIFI_STA_STOP:
            Serial.println("WiFi clients stopped");
            break;
        case ARDUINO_EVENT_WIFI_STA_CONNECTED:
            Serial.println("Connected to access point");
            break;
    }
}

```

```
case ARDUINO_EVENT_WIFI_STA_DISCONNECTED:
    Serial.println("Disconnected from WiFi access point");
    wifi_state = 0;
    reconnect_count = 10;
    break;
case ARDUINO_EVENT_WIFI_STA_AUTHMODE_CHANGE:
    Serial.println("Authentication mode of access point has changed");
    break;
case ARDUINO_EVENT_WIFI_STA_GOT_IP:
    Serial.print("Obtained IP address: ");
    Serial.println(WiFi.localIP());
    wifi_state = 2;
    break;
case ARDUINO_EVENT_WIFI_STA_LOST_IP:
    Serial.println("Lost IP address and IP address is reset to 0");
    break;
case ARDUINO_EVENT_WPS_ER_SUCCESS:
    Serial.println("WiFi Protected Setup (WPS): succeeded in enrollee mode");
    break;
case ARDUINO_EVENT_WPS_ER_FAILED:
    Serial.println("WiFi Protected Setup (WPS): failed in enrollee mode");
    break;
case ARDUINO_EVENT_WPS_ER_TIMEOUT:
    Serial.println("WiFi Protected Setup (WPS): timeout in enrollee mode");
    break;
case ARDUINO_EVENT_WPS_ER_PIN:
    Serial.println("WiFi Protected Setup (WPS): pin code in enrollee mode");
    break;
case ARDUINO_EVENT_WIFI_AP_START:
    Serial.println("WiFi access point started");
    break;
case ARDUINO_EVENT_WIFI_AP_STOP:
    Serial.println("WiFi access point stopped");
    break;
case ARDUINO_EVENT_WIFI_AP_STACONNECTED:
    Serial.println("Client connected");
    break;
case ARDUINO_EVENT_WIFI_AP_STADISCONNECTED:
    Serial.println("Client disconnected");
    break;
case ARDUINO_EVENT_WIFI_AP_STAIPASSIGNED:
    Serial.println("Assigned IP address to client");
    break;
case ARDUINO_EVENT_WIFI_AP_PROBEREQRECVED:
    Serial.println("Received probe request");
    break;
case ARDUINO_EVENT_WIFI_AP_GOT_IP6:
    Serial.println("AP IPv6 is preferred");
    break;
case ARDUINO_EVENT_WIFI_STA_GOT_IP6:
    Serial.println("STA IPv6 is preferred");
    break;
case ARDUINO_EVENT_ETH_GOT_IP6:
    Serial.println("Ethernet IPv6 is preferred");
    break;
case ARDUINO_EVENT_ETH_START:
    Serial.println("Ethernet started");
    break;
case ARDUINO_EVENT_ETH_STOP:
    Serial.println("Ethernet stopped");
    break;
```



```

    case ARDUINO_EVENT_ETH_CONNECTED:
        Serial.println("Ethernet connected");
        break;
    case ARDUINO_EVENT_ETH_DISCONNECTED:
        Serial.println("Ethernet disconnected");
        break;
    case ARDUINO_EVENT_ETH_GOT_IP:
        Serial.println("Obtained IP address");
        break;
    default: break;
}
}

void wifiSetup() {
    WiFi.onEvent(onWiFiEvent);
}

#endif

```

4.3 wlan.h.in

```

#ifndef _WLAN_H_
#define _WLAN_H_

// kopieren in Datei wlan.h
// W-Lan-Zugang einrichten
const char* ssid = "HTL-IoT";
const char* password = "password";
// ntpServer from DHCP will overwrite this one
const char* ntpServer = "10.10.204.254";
// htl: "10.10.204.254";
// at.pool.ntp.org: 151.236.30.71
// TimeZone rule for Europe/Vienna including daylight adjustment rules
const char* time_zone = "CET-1CEST-2,M3.5.0/02:00:00,M10.5.0/03:00:00";

#endif

```