```
/***************************************************
* NTP2DCF77 using ESP8266
* Thanks to
*   NTP time with > 100ms precision, thanks ddrown
https://github.com/ddrown/Arduino_NTPClient
*   OverTheAir programming possible (whwn OTA flag set)
*   WifiManager, thanks to tzapu https://github.com/tzapu/WiFiManager
*   DCF77 pulse generation inspired by Raspberry Pi version from hzeller
https://github.com/hzeller/txtempus
*   Excellent info on how to add an antenna to the raspberry
https://www.youtube.com/watch?v=6SHGAEhnsYk
*   And many more to thank ...
*
* WARNING: Check if it is allowed in your country to genereate this (very
low level) RF signal. Keep this signal as low as possible , in the cm
range from your clock.
*
* Tested on a Wemos (Lolin) D1 mini
*
* From https://github.com/tzapu/WiFiManager:
*   when your ESP starts up, it sets it up in Station mode and tries to
connect to a previously saved Access Point
*   if this is unsuccessful (or no previous network saved) it moves the
ESP into Access Point mode and spins up a DNS and WebServer (default ip
192.168.4.1)
*   using any wifi enabled device with a browser (computer, phone,
tablet) connect to the newly created Access Point
*   because of the Captive Portal and the DNS server you will either get
a 'Join to network' type of popup or get any domain you try to access
redirected to the configuration portal
*   choose one of the access points scanned, enter password, click save
*   ESP will try to connect. If successful, it relinquishes control back
to your app. If not, reconnect to AP and reconfigure.
*
* D8 (GPIO15): I2S Clock is 'misused' as a stable 77.5KHz carrier (fixed
pin) Max 12mA drive
* D2 (GPIO04): Is used as modulation pin Max 20mA sink
* D4 I2S uses this pin for the WS signal, not used here, but on the Wemos
it is connected to the led, so it will glow. Did not test if writing to
this ledport kills the I2S clk function.
* D8 and D2 must be connected together with 2 resistors to get a 85%
modulation. I use these values (maybe you have to change values when
connecting a ferrite antenna, or add a transistor):
* D8 ---- 620 ohm -----+----- 68 ohm ---- D2
*                      |-------------------------------- Antenna (or
open wire)
```

```
* See Andreas YT-channel for experimenting with antenna's
https://www.youtube.com/watch?v=6SHGAEhnsYk, but a single turn of wire
around/close to the receiving antenna can be sufficient.
*
* Timezone and WT/ST config can be changed in settings.h (Most clocks are
needing a reset when already sync'ed to another time)
*
* Improvements: It would be nice to have a PDM 77,5KHz, 100% and 15%)
signal out of I2S data port. Sinus is better than square wave, and no
need for the modulation port.
*
*
* Version 1.0 EVB 19-10-2019 initial
***************************************************/


#include <ESP8266WiFi.h> // https://github.com/esp8266/Arduino
#include <WiFiManager.h> // https://github.com/tzapu/WiFiManager
#include <ArduinoOTA.h>  // Over The Air SW download
#include <Clock.h>       // adapted millisecond precision clock library:
https://github.com/ddrown/Arduino_Clock
#include <Timezone.h>    // Timezone library ported to ESP8266:
https://github.com/ddrown/Timezone
#include <NTPClient.h>   // adapted from original NTPClient, ESP8266 NTP
client with millisecond precision:
https://github.com/ddrown/Arduino_NTPClient
#include <ClockPID.h>    // PID controller, ClockPID calculates the rate
adjustment the local clock needs based on offset measurements.
https://github.com/ddrown/Arduino_ClockPID
#include "settings.h"    // see this file to change the settings
#include <i2s.h>         // I2S for 77.5KHz generation
#include <core_esp8266_timer.cpp>


#define VERSION 1.0
#define SERIAL_SPEED 115200
#define LOCAL_NTP_PORT 2389 // local port to listen for UDP packets
#define SERIALPORT Serial
#define MODULATION_PORT D2 //gpio4. Don't use I2S ports. gpio16 doesn't
work in open drain mode


bool ota_flag = false; //set it to true for Over The air SW download
long temp_millis;
uint32_t ntp_packets = 0;
uint8_t prevsec = 99;
```

```
time_t local, next_ntp, fast_ntp_done;
const char *(weekdays[7]) = {"Sunday", "Monday", "Tuesday", "Wednesday",
"Thursday", "Friday", "Saturday"};
struct timems startTS;
struct timems nowTS;
struct timems last_t;
struct timems beforeNTP, afterNTP;
TimeChangeRule *tcr;


ESP8266WebServer server(80);
WiFiManager wifiManager;
IPAddress timeServerIP;
NTPClient ntp;
WiFiClient client;


void ICACHE_RAM_ATTR timerISR()
{
  digitalWrite(MODULATION_PORT,HIGH);
}


uint64_t to_bcd(uint8_t n) {
  return ((n / 10) << 4) | (n % 10);
}


uint64_t parity(uint64_t d, uint8_t from, uint8_t to_including) {
  uint8_t result = 0;
  for (int bit = from; bit <= to_including; ++bit) {
    if (d & (1ULL << bit)) result++;
  }
  return result & 0x1;
}


void ntp_setup()
{
  SERIALPORT.println("Starting NTP/UDP");
  ntp.begin(LOCAL_NTP_PORT);
  WiFi.hostByName(ntpServerName, timeServerIP); // TODO: lookup failures
and re-lookup DNS pools
  SERIALPORT.print("NTP IP: ");
  SERIALPORT.println(timeServerIP.toString());
}
```

```
int ntp_loop(bool ActuallySetTime)
{
  PollStatus NTPstatus;
  static struct timems startLocalTS = {.tv_sec = 0};
  static struct timems startRemoteTS = {.tv_sec = 0};
  int retval = 0;


  ntp.sendNTPpacket(timeServerIP);
  while ((NTPstatus = ntp.poll_reply(ActuallySetTime)) == NTP_NO_PACKET)
  {           // wait until we get a response
    delay(1); // allow the background threads to run
  }
  if (NTPstatus == NTP_TIMEOUT)
  {
    adjustClockSpeed_ppm(ClockPID.d_out());
    SERIALPORT.println("NTP client timeout, using just the last D
sample");
    retval = -1;
  }
  else if (NTPstatus == NTP_PACKET)
  {
    struct timems nowTS, remoteTS;
    uint32_t ms_delta;
    int32_t ppm_error;


    now_ms(&nowTS);
    ntp_packets++;
    ms_delta = nowTS.raw_millis - startLocalTS.raw_millis;
    ntp.getRemoteTS(&remoteTS);
    if ((startLocalTS.tv_sec == 0) || (ms_delta > 2140000000))
    { // reset clock on startup and when it gets close to wrapping at
2^31
      startLocalTS.tv_sec = nowTS.tv_sec;
      startLocalTS.tv_msec = nowTS.tv_msec;
      startLocalTS.raw_millis = nowTS.raw_millis;
      startRemoteTS.tv_sec = remoteTS.tv_sec;
      startRemoteTS.tv_msec = remoteTS.tv_msec;
      startRemoteTS.raw_millis = remoteTS.raw_millis;
      ClockPID.reset_clock();
      SERIALPORT.println("Reset Clock");
      ms_delta = 0;
    }
    int32_t offset = ntp.getLastOffset_RTT();
    int32_t raw_offset = ts_interval(&startRemoteTS, &remoteTS) -
ms_delta;
    uint32_t rtt = ntp.getLastRTT();
    float clock_error = ClockPID.add_sample(ms_delta, raw_offset,
offset);


    adjustClockSpeed_ppm(clock_error);


    SERIALPORT.print("now=");
    SERIALPORT.print(nowTS.tv_sec);
    SERIALPORT.print(" rtt=");
    SERIALPORT.print(rtt);
    SERIALPORT.print(" ppm=");
    SERIALPORT.print(clock_error * 1000000);
    SERIALPORT.print(" offset=");
    SERIALPORT.print(offset);
    SERIALPORT.print(" raw=");
    SERIALPORT.print(raw_offset);
    SERIALPORT.print(" delta=");
    SERIALPORT.println(ms_delta);
    SERIALPORT.print(ClockPID.p());
    SERIALPORT.print(",");
    SERIALPORT.print(ClockPID.i());
    SERIALPORT.print(",");
    SERIALPORT.print(ClockPID.d() * 1000000);
    SERIALPORT.print(",");
    SERIALPORT.print(ClockPID.d_chi() * 1000000);
    SERIALPORT.print(",");
    SERIALPORT.print(ClockPID.out() * 1000000);
    SERIALPORT.print(",");
    SERIALPORT.print(ClockPID.p_out() * 1000000);
    SERIALPORT.print(",");
    SERIALPORT.print(ClockPID.i_out() * 1000000);
    SERIALPORT.print(",");
    SERIALPORT.println(ClockPID.d_out() * 1000000);
  }
  return retval;
}


void configModeCallback(WiFiManager *myWiFiManager)
{
  SERIALPORT.println("Entered config mode");
  SERIALPORT.println(WiFi.softAPIP());
  //if you used auto generated SSID, print it
  SERIALPORT.println(myWiFiManager->getConfigPortalSSID());
  SERIALPORT.println("No WIFI configuration, select WIFI netwerk
'NTP2DCF77' / IP: 192.168.4.1");
}


void calculate_dcf77_pulses(void){
  static uint64_t pulse_train = 0x100000; //bit 20 = 1
  uint64_t dow = 1;
  uint8_t current_second;
  static bool pulsetrain_valid = false;

  //a new second when we are here
  current_second = second(local);
  if (current_second == 59){
    digitalWrite(MODULATION_PORT,HIGH);
    pulsetrain_valid = true; //(will be valid in the next second for all
seconds anyway)
    SERIALPORT.print("************   ");
  }else if (pulsetrain_valid == true){ //wait at startup for correct data
before modulation
    digitalWrite(MODULATION_PORT,LOW); //every second starts with a
modulation, except sec 59
    if ((pulse_train & (1ULL << current_second )) > 0){
      timer1_write(1000000); // 200ms
      SERIALPORT.print("============   ");
    }else{
      timer1_write(500000); // 100ms
      SERIALPORT.print("======         ");
    }
  }
  if (current_second == 0){
    pulse_train = 0x100000; //bit 20 = 1
    //local+60, we need next minute info
    if (TIMEZONE.locIsDST(local+60) == true)
      pulse_train |= (1ULL << 17); //ST
    else
           pulse_train |= (1ULL << 18); //WT
    pulse_train |= to_bcd(minute(local+60)) << 21;
    pulse_train |= parity(pulse_train, 21, 27) << 28;
    pulse_train |= to_bcd(hour(local+60)) << 29;
    pulse_train |= parity(pulse_train, 29, 34) << 35;
    pulse_train |= to_bcd(day(local+60)) << 36;
    dow = to_bcd(weekday(local+60));
    (dow == 1) ? dow = 7 : dow--; //dcf77 starts on monday iso sunday
    pulse_train |= dow << 42;
    pulse_train |= to_bcd(month(local+60)) << 45;
    pulse_train |= to_bcd(year(local+60)-2000) << 50;
    pulse_train |= parity(pulse_train, 36, 57) << 58;
  }
}


void setup(){
  pinMode(MODULATION_PORT,OUTPUT_OPEN_DRAIN);
  digitalWrite(MODULATION_PORT,HIGH);
  SERIALPORT.begin(SERIAL_SPEED); //must be before i2s_begin, otherwise
it kills chip dma config. Only tx is available for serial comm
  while (!SERIALPORT);
  SERIALPORT.println("startup NTP2DCF77");
  SERIALPORT.println("Version: " + String(VERSION));
  //wifiManager.resetSettings(); // clears pw data, only for test
  wifiManager.setAPCallback(configModeCallback); //function called when
entering config AP
  //try to connect to wifi network, if failed startup AP with config menu
ssid/pw
  wifiManager.setTimeout(180);
  if (!wifiManager.autoConnect("NTP2DCF77")){
    SERIALPORT.println("Failed to connect and hit timeout");
    delay(3000);
    //reset and try again
    ESP.reset();
    delay(5000);
  }
  else
    SERIALPORT.printf("\nConnected successful with wifiManager to SSID
'%s'\n", WiFi.SSID().c_str());
  if (ota_flag){
    //next start up OTA software download with Arduino UI
    ArduinoOTA.onStart([]() { SERIALPORT.println("Start OTA update"); });
    ArduinoOTA.onEnd([]() { SERIALPORT.println("End OTA update"); });
    ArduinoOTA.onProgress([](unsigned int progress, unsigned int total) {
SERIALPORT.printf("Progress: %u%%\r", (progress / (total / 100))); });
    ArduinoOTA.onError([](ota_error_t error)
{SERIALPORT.printf("Error[%u]: ", error);
    if (error == OTA_AUTH_ERROR) SERIALPORT.println("Auth Failed");
    else if (error == OTA_BEGIN_ERROR) SERIALPORT.println("Begin
Failed");
    else if (error == OTA_CONNECT_ERROR) SERIALPORT.println("Connect
Failed");
```

```
    else if (error == OTA_RECEIVE_ERROR) SERIALPORT.println("Receive
Failed");
    else if (error == OTA_END_ERROR) SERIALPORT.println("End Failed");
});
    ArduinoOTA.begin();
    SERIALPORT.print("OTA Ready, IP address: ");
    SERIALPORT.println(WiFi.localIP());
  }
  i2s_begin();
  i2s_set_rate(2422); //this is close to 77500Hz
  timer1_attachInterrupt(timerISR);
  timer1_enable(TIM_DIV16, TIM_EDGE, TIM_SINGLE);


  ntp_setup();
  SERIALPORT.println("Delayed start.");
  delay(1000);
  last_t.tv_sec = 0;
  last_t.tv_msec = 0;
  while (ntp_loop(true) < 0)
  { // set time
    SERIALPORT.println("NTP request failed, retrying");
    delay(64000);
  }
  next_ntp = now() + 64;
  fast_ntp_done = now() + 64 * 4; // first 4 samples at 64s each
  now_ms(&startTS);
  temp_millis = millis();
}


void loop(void){
  if (ota_flag){
    SERIALPORT.println("Start OTA window");
    while ((millis() - temp_millis) < 30000)
    { //30 sec time windows after a reboot to start OTA update
      ArduinoOTA.handle();
      yield();
    }
    ota_flag = false;
    SERIALPORT.println("End OTA window");
  }
  yield();
  now_ms(&nowTS);
  if (nowTS.tv_sec != last_t.tv_sec){
    // new second
    now_ms(&last_t);
    local = TIMEZONE.toLocal(last_t.tv_sec, &tcr);
    if (second(local) != prevsec){ //don't retrig timer when adj internal
clock
      prevsec = second(local);
      calculate_dcf77_pulses();
      SERIALPORT.println(hour(local) + String(" ") + minute(local) +
String(" ") + second(local) + String("  ") +
String(weekdays[weekday(local)-1]) + String(" ---- ") + day(local) +
String(" ") + month(local) + String(" ") + year(local));
    }
    // repoll on seconds not ending in 0
    if ((last_t.tv_sec > next_ntp) && ((second(local) % 10) != 0)){
      now_ms(&beforeNTP);
      ntp_loop(false);
      if (last_t.tv_sec > fast_ntp_done)
        next_ntp = last_t.tv_sec + NTP_INTERVAL;
      else
        next_ntp = last_t.tv_sec + 64;
      now_ms(&afterNTP);
      SERIALPORT.print("ntp took ");
      SERIALPORT.print(ts_interval(&beforeNTP, &afterNTP));
      SERIALPORT.println("ms");
    }
  }
}
```