Arduino Projects

The two projects I set myself to learn Arduino IDE include the following:

- Build a clock
- Create a video player

These projects each had their own set of challenges which I will outline in the proceeding document.

Build a Clock

Goals

- ☐ Must Show Time (accurately)
- ☐ Display Temperature (local)
- ☐ Display Up to Milliseconds

Accurate Timing and Milliseconds

After conducting research on the Arduino I decided to use for the projects (ESP8266) I noticed some users complaining about the clock rate slowing down over time. This ruled out using the board offline unless I wanted to purchase a separate clock for time keeping. To address the hurdle, I purchased a Wifi capable ESP8266 which would allow me to sync up with an API.

Considering I want to continually sync with the API (which is only accurate to the minutes) without lagging the display, I mapped out how I could go about this without desynching too much.

Possible Solutions

1) Sync once on startup
2) Sync every minute
3) Sync more than every minute
4) Sync every second

Each of these solutions has various benefits and detriments. The solution I went with mixes methods 3 and 4.

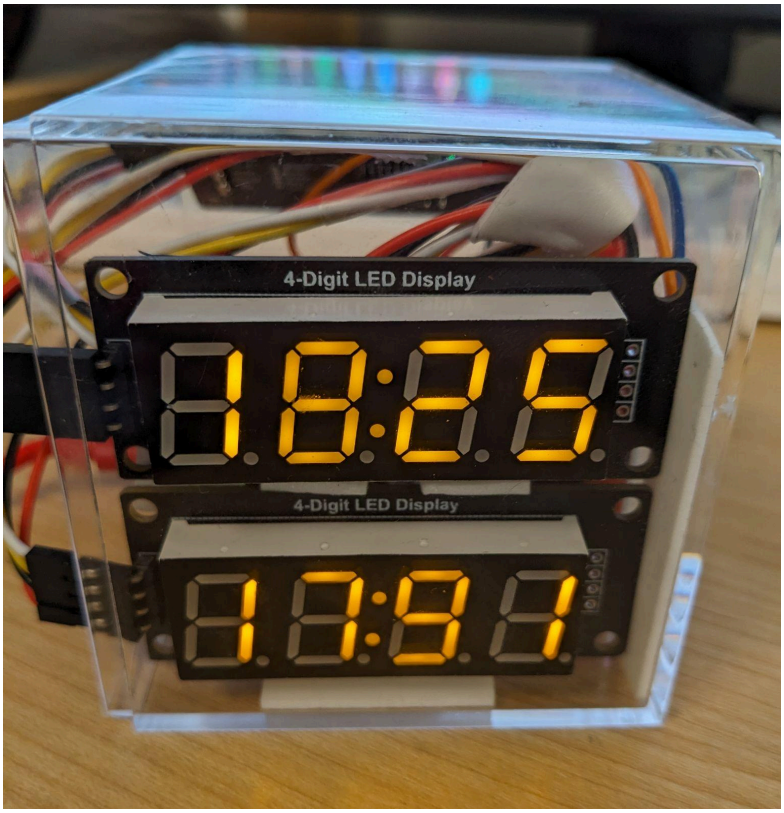
If we want the clock to best line up with the real time, it is important that it syncs whenever an update to the API's time takes place. Upon startup, the Arduino makes a call to the API every half second until the value it receives is first updated. At this point, the device is within half a second of the API's time meaning we can now use the device's own clock for the bulk of the timing.

However, to avoid desynch over long periods of usage, method 3 comes into play. If we decide to call from the API each minute, it is possible that we encounter a scenario where our clock skips over a minute. This would look quite strange since we are displaying down to the millisecond. If instead we check every 30 seconds, there at most will be half a minute desynch and once the clock reaches 30 seconds of the minute it will be updated to the start accordingly.

In my testing, the internal clock seems quite stable where I have not noticed this 30 second case occur, but regardless of how good the clock is, it is inevitable if left on long enough.

Display Temperature

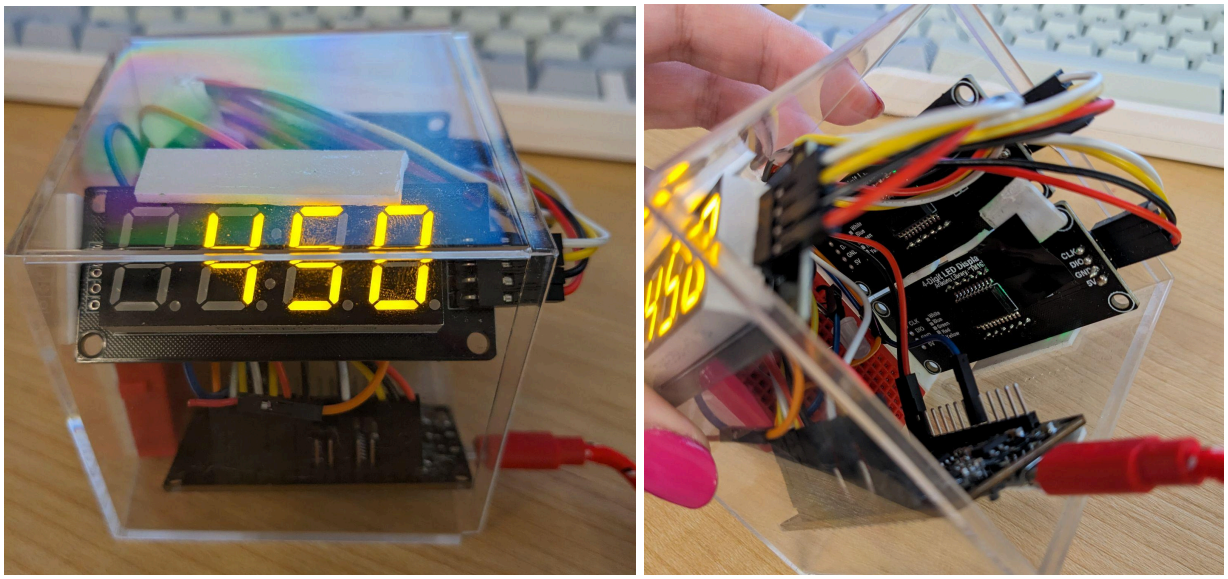
Displaying the temperature simply requires an API call from a server local to the area. Since the weather changes infrequently I sample this once every five minutes to reduce the memory usage during clock updates.



Main Clock Display

Top – Hours, Minutes

Bottom – Seconds, Centiseconds



Temperature F (3 place accurate)

Clock Code .ino

```
#include <ESP8266WiFi.h>
#include <NTPClient.h>
#include <WiFiUdp.h>
#include <TM1637Display.h>
#include <ArduinoJson.h>
#include <ESP8266HTTPClient.h>
#include <U8g2lib.h>

// WiFi credentials
const char* ssid = "GWconnect";
const char* password = "";

// NTP Client setup
WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, "pool.ntp.org");

// Global variables for weather update
unsigned long lastWeatherUpdate = 0;
const long weatherUpdateInterval = 300000; // 5 minutes in milliseconds
String storedTemperature = ""; // Store the fetched temperature here
unsigned long lastNTPUpdateMillis2 = 0;

// Display setups
#define TX 1
#define D1 5
#define D2 4
#define D3 0
#define D4 2
#define D7 13

#define LED_PIN 16 // D0 - GPIO16


int CLK1 = D7;
int DIO1 = TX;
```

```
TM1637Display display1(CLK1, DIO1); //secs

int CLK2 = D3;
int DIO2 = D4;
TM1637Display display2(CLK2, DIO2); //mins

int CLK3 = D1;
int DIO3 = D2;
TM1637Display display3(CLK3, DIO3); //temp

int updatefreq = 500;

unsigned long lastNTPUpdateMillis = 0;

void setup() {
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(updatefreq);
  }

  pinMode(LED_PIN, OUTPUT);

  timeClient.begin();
  timeClient.setTimeOffset(-5 * 3600);
  while(!timeClient.update()) {
    timeClient.forceUpdate();
    delay(updatefreq);
  }

  lastNTPUpdateMillis = millis();
  displayHoursAndMinutes();

  display1.setBrightness(0x0f);
  display1.clear();
  display2.setBrightness(0x0f);
  display2.clear();
  display3.setBrightness(0x0f);
  display3.clear();

  // Fetch the initial temperature
```

```
  storedTemperature = getWeatherData();
  displayTemperature(storedTemperature);

}

int startup = 0;
unsigned long lastLedToggle = 0;

#define rate 25
#define LED_BLINK_INTERVAL 1000

void loop() {
  unsigned long currentMillis = millis();

  // Update weather data every 5 minutes
  if (currentMillis - lastWeatherUpdate >= weatherUpdateInterval || startup == 0) {
    lastWeatherUpdate = currentMillis;
    storedTemperature = getWeatherData();  // Update the stored temperature
  }

  if (currentMillis - lastNTPUpdateMillis >= updatefreq || startup == 0) {
    lastNTPUpdateMillis = currentMillis;
    displayHoursAndMinutes();
    displayTemperature(storedTemperature);  // Display the stored temperature
  }

  if (currentMillis - lastLedToggle >= LED_BLINK_INTERVAL) {
    lastLedToggle = currentMillis;
    digitalWrite(LED_PIN, !digitalRead(LED_PIN)); // Toggle LED state
  }

  displaySecondsAndCentiseconds();
  delay(rate);
  startup++;
}

// Function to fetch weather data
String getWeatherData() {
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;
```

```
    WiFiClient client;
    String apiKey = "c…1";
        String  url  =  "http://api.weatherapi.com/v1/current.json?key="  +  apiKey  +
"&q=20052&aqi=no";
    http.begin(client, url);
    int httpCode = http.GET();

    if (httpCode > 0) {
      String payload = http.getString();
      http.end();

      DynamicJsonDocument doc(1024);
      deserializeJson(doc, payload);

      String temperature = String(doc["current"]["temp_f"].as<float>());
      return temperature;
    } else {
      http.end();
      return "Error";
    }
  }
  return "No WiFi";
}

int lastKnownHour = -1;
int lastKnownMinute = -1;
int countcheck = 0;

void displayHoursAndMinutes() {
  time_t rawTime = timeClient.getEpochTime();
  struct tm *ptm = localtime(&rawTime);

  int hours = ptm->tm_hour;
  int minutes = ptm->tm_min;

  // Check if hours or minutes have changed
  if (hours != lastKnownHour || minutes != lastKnownMinute) {
    lastNTPUpdateMillis2 = millis();  // Reset the seconds timer
    lastKnownHour = hours;
    lastKnownMinute = minutes;
```

```
    countcheck++;
  }

  if (countcheck > 1) {
    updatefreq = 30000;
  }

  display2.showNumberDecEx(hours, 0b01000000, true, 2, 0); // Display hours
  display2.showNumberDecEx(minutes, 0b00000000, true, 2, 2); // Display minutes
}

void displaySecondsAndCentiseconds() {
  unsigned long currentTime = millis();
  unsigned long timeElapsedSinceUpdate = (currentTime - lastNTPUpdateMillis2);
  int seconds = (timeElapsedSinceUpdate / 1000) % 60;
  int centiseconds = (timeElapsedSinceUpdate % 1000) / 10;
  display1.showNumberDecEx(seconds, 0b01000000, true, 2, 0);
  display1.showNumberDecEx(centiseconds, 0, true, 2, 2);
}

void displayTemperature(String temperature) {
  float tempValue = temperature.toFloat();

  // Extract the integer part and the first decimal digit
  int intPart = (int)tempValue; // Integer part
  int decimalPart = (int)(tempValue * 10) % 10; // First decimal digit

  if (intPart >= 100) {
    // If the integer part is 100 or more, display all digits
    display3.showNumberDecEx(intPart, 0b00000000, true, 3, 0);
    display3.showNumberDecEx(decimalPart, 0b10000000, false, 1, 3);
  } else {
    // If the integer part is less than 100, leave the first digit blank
     display3.showNumberDecEx(intPart, 0b00000000, false, 2, 1); // Display on second
and third segments
      display3.showNumberDecEx(decimalPart, 0b10000000, false, 1, 3); // Display on
fourth segment
  }
}
```

Create a Video Player

Goals

- ☐ Play Video on Binary OLED
- ☐ Full Length (3+ mins)
- ☐ Non Specific Solution

Play Video

When deciding this endeovor, it is important to note the device I'm working with. The ESP8266 wifi with OLED has a 128x64 display, 80 KB of usable RAM, and 1 MB flash memory. At first I wanted to use the flash because I could just encode each frame into binary so the 1 MB should have been enough. However this would be a difficult process if I ever wanted to use a different video or not corrupt my device.

Instead I decided on a dynamic method of running the video. Using python I wrote a script to convert an mp4 into 1-bit-depth bitmap images which I then uploaded to github. This would allow each frame to be a very small and consistent packet size which I could still view for debugging purposes. I tried other methods for compression such as only binary and concatonating the frame data with 8-bit data, but the bitmap still provided the most benefits.

The video I decided to use was the music video *Bad Apple!!* from the Touhou Project series. This video has been used for projects from others online, inspiring me to do the same. Not only is the music video only black and white, but it is also standard definition avoiding two drawbacks to the device. These are:

1) Binary Display – Pixels can only be on or off
2) Small RAM – Frames need to have minimal data

Once I decided upon my approach, there were a lot of hurdles to overcome displaying the images properly. The noteworthy of which include:

- How do I display a frame on ESP8266
- What is the best way to retrieve the frames from github (bulk or stream)
- What is limiting framerate (RAM or github)
- How to convert bitmap to displayable frame
- What is the minimum memory I can use for each frame
- Which processes should be in compile vs runtime

The only problem I'll discuss in more depth is how I got each frame to display. I discovered that for the ESP8266 there was an easy function to display images in XBM data type. However, XBM is much larger than binary so it was important to convert to it during the runtime, instead of when uploading to github. When converting each frame to XBM, the bitmap header must be removed and each binary pixel set to the corresponding hex value. At one point I had the header read and filtered out, but then I realized I could just read however many bits the header is into a garbage array I called "buffer". This significantly reduced the runtime since the header is quite small and keeping it constrained to simply read straight into a garbage array was very efficient.

Another thing to note is at one point I skip a certain number of frames to get the framerate of the video to look correct. In this case, 1 in every 3 frames is being read, however this value is easily adjustable.

Full Length Video / Non Specific

By deciding to use github and streaming frame by frame to the device, any length video could be played as long as I run the mp4 through the python script I mentioned earlier. This not only means that any length video, but any video I convert to 1-bit-depth bitmap within the 128x64 could be played as long as I adjust the parameters in the header file.

Video Header .h

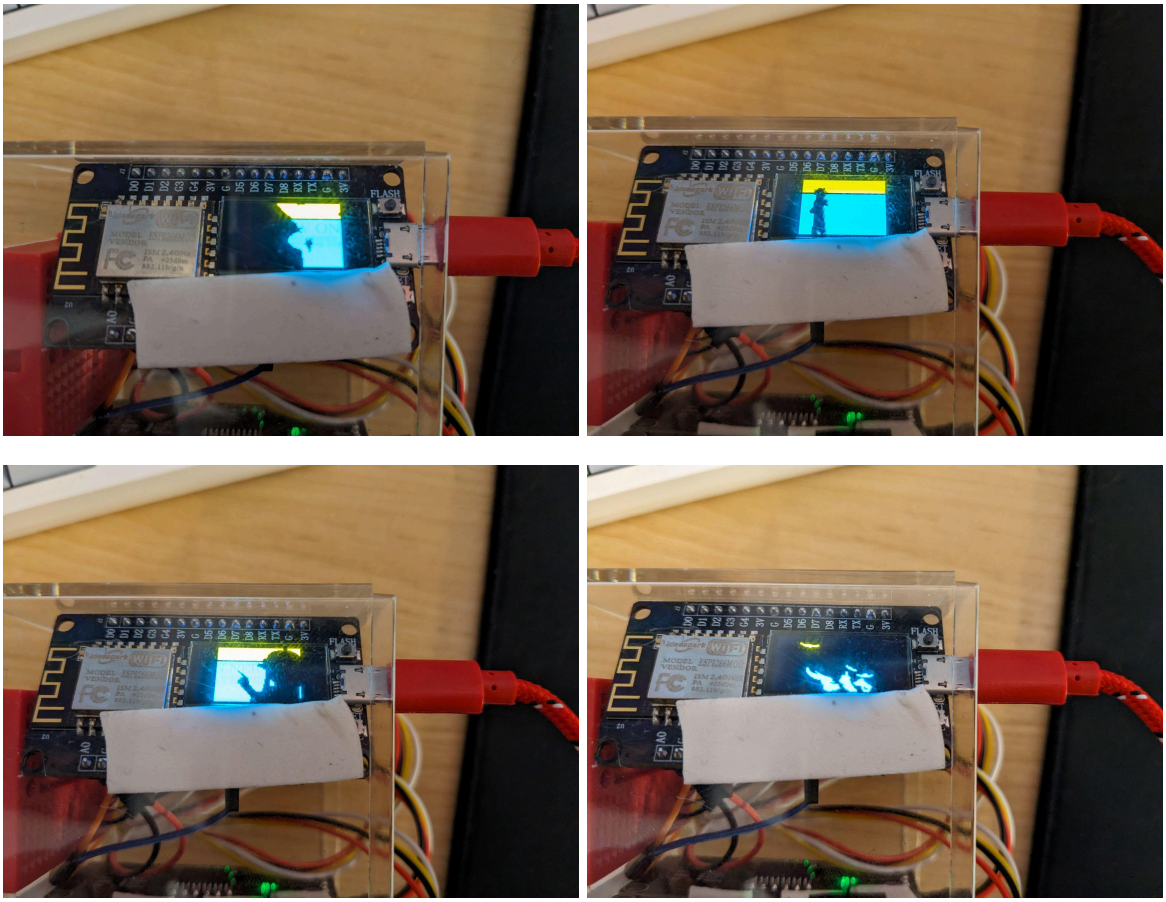
```
#ifndef BAD_APPLE_H
#define BAD_APPLE_H

#include <ESP8266WiFi.h>
#include <WiFiClientSecure.h>
#include <ESP8266HTTPClient.h>
#include <Wire.h>
#include <SSD1306Wire.h>
#include <string>

#define SCREEN_WIDTH 128  // OLED display width, change according to your display
#define HEADER_SIZE 62
#define IMAGE_WIDTH 86
#define HEIGHT 64
#define FILE_SIZE 768
#define LINE_SIZE 12

SSD1306Wire display(0x3c, 12, 14);  // ADDRESS, SDA, SCL

WiFiClientSecure wifiClient;
HTTPClient http;

bool* img = new bool[IMAGE_WIDTH * HEIGHT];
unsigned char* buffer = new unsigned char[HEADER_SIZE];
unsigned char* data = new unsigned char[FILE_SIZE];

// Constants for frame handling
short frameNumber = 1; // Starting frame

#endif // BAD_APPLE_H
```

Video Code .ino

```
#include "badapple.h"

void setup() {

  Serial.begin(115200);

  WiFi.begin("GWconnect", "");
  Serial.print("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nConnected to WiFi");

  display.init();
  display.flipScreenVertically();
  wifiClient.setInsecure();

  memset(img, 0, sizeof(img));
  memset(data, 0, sizeof(data));

  String url = "https://raw.githubusercontent.com/drewreno/badappbmp/fullbmp";

  http.begin(wifiClient, url);
  delay(1000);
  Serial.println("starting");
  delay(1000);
}

void loop() {
  Main();
  frameNumber += 3;
}

void Main() {
  memset(data, 0, sizeof(data));
```

```
   String url = "https://raw.githubusercontent.com/drewreno/badappbmp/fullbmp/frame_"
+ String(frameNumber) + ".bmp";

  http.begin(wifiClient, url);
  unsigned char httpCode = http.GET();

  if (httpCode == HTTP_CODE_OK) {
    WiFiClient *stream = http.getStreamPtr();

    /*Read BMP header
    unsigned char head[HEADER_SIZE];
      short w = head[18] + (((short)head[19]) << 8) + (((short)head[20]) << 16) +
(((short)head[21]) << 24);
      short h = head[22] + (((short)head[23]) << 8) + (((short)head[24]) << 16) +
(((short)head[25]) << 24);
    short lineSize = ((w + 31) / 32) * 4;
    short fileSize = lineSize * h;*/

    stream->readBytes(buffer, HEADER_SIZE);

    stream->readBytes(data, FILE_SIZE);

    for (unsigned char j = 0, rev_j = HEIGHT - 1; j < HEIGHT; j++, rev_j--) {
      for (unsigned char i = 0; i < IMAGE_WIDTH; i++) {
        unsigned char byte_ctr = i / 8;
        unsigned char data_byte = data[j * LINE_SIZE + byte_ctr];
        short pos = rev_j * IMAGE_WIDTH + i;
        unsigned char mask = 0x80 >> (i % 8);
        img[pos] = (data_byte & mask) ? 1 : 0;
      }
    }

    memset(data, 0, sizeof(data));

    //convert to xbm
    for (short y = 0; y < HEIGHT; y++) {
      for (short x = 0; x < IMAGE_WIDTH; x++) {
        short pos = y * IMAGE_WIDTH + x;
        short xbmPos = y * ((IMAGE_WIDTH + 7) / 8) + x / 8;
```

```
        if (img[pos] == 1) {
          data[xbmPos] |= (1 << (x % 8));
        } else {
          data[xbmPos] &= ~(1 << (x % 8));
        }
      }
    }

    memset(img, 0, sizeof(img));

    //display
    display.clear();
    display.drawXbm(21, 0, IMAGE_WIDTH, HEIGHT, data);
    display.display();
    http.end();
  }
}
```



Frames from Video