#include <SoftwareSerial.h>

#include <SPI.h>

#include <SD.h>

#include <Adafruit\_VS1053.h>

// These are the pins used

#define VS1053\_RESET -1 // VS1053 reset pin (not used!)

// Feather ESP8266

// Feather M4, M0, 328, ESP32S2, nRF52840 or 32u4

#define VS1053\_CS 6 // VS1053 chip select pin (output)

#define VS1053\_DCS 10 // VS1053 Data/command select pin (output)

#define CARDCS 5 // Card chip select pin

// DREQ should be an Int pin \*if possible\* (not possible on 32u4)

#define VS1053\_DREQ 9 // VS1053 Data request, ideally an Interrupt pin

Adafruit\_VS1053\_FilePlayer musicPlayer =

Adafruit\_VS1053\_FilePlayer(VS1053\_RESET, VS1053\_CS, VS1053\_DCS, VS1053\_DREQ, CARDCS);

#include "FastLED.h"

const int buttonPin = 11;

int counter = 0;

#define NUM\_LEDS 216

#define LED\_TYPE WS2811

#define COLOR\_ORDER GRB

#define DATA\_PIN 2

//#define CLK\_PIN 4

#define VOLTS 12

#define MAX\_MA 4000

CRGBArray<NUM\_LEDS> leds;

// Overall twinkle speed.

// 0 (VERY slow) to 8 (VERY fast).

// 4, 5, and 6 are recommended, default is 4.

#define TWINKLE\_SPEED 4

// Overall twinkle density.

// 0 (NONE lit) to 8 (ALL lit at once).

// Default is 5.

#define TWINKLE\_DENSITY 6

// How often to change color palettes.

#define SECONDS\_PER\_PALETTE 30

// Also: toward the bottom of the file is an array

// called "ActivePaletteList" which controls which color

// palettes are used; you can add or remove color palettes

// from there freely.

// Background color for 'unlit' pixels

// Can be set to CRGB::Black if desired.

CRGB gBackgroundColor = CRGB::Black;

// Example of dim incandescent fairy light background color

// CRGB gBackgroundColor = CRGB(CRGB::FairyLight).nscale8\_video(16);

// If AUTO\_SELECT\_BACKGROUND\_COLOR is set to 1,

// then for any palette where the first two entries

// are the same, a dimmed version of that color will

// automatically be used as the background color.

#define AUTO\_SELECT\_BACKGROUND\_COLOR 0

// If COOL\_LIKE\_INCANDESCENT is set to 1, colors will

// fade out slighted 'reddened', similar to how

// incandescent bulbs change color as they get dim down.

#define COOL\_LIKE\_INCANDESCENT 1

CRGBPalette16 gCurrentPalette;

CRGBPalette16 gTargetPalette;

void setup() {

Serial.begin(115200);

// if you're using Bluefruit or LoRa/RFM Feather, disable the radio module

//pinMode(8, INPUT\_PULLUP);

// Wait for serial port to be opened, remove this line for 'standalone' operation

//while (!Serial) { delay(1); }

delay(500);

Serial.println("\n\nAdafruit VS1053 Feather Test");

if (! musicPlayer.begin()) { // initialise the music player

Serial.println(F("Couldn't find VS1053, do you have the right pins defined?"));

while (1);

}

Serial.println(F("VS1053 found"));

musicPlayer.sineTest(0x44, 5); // Make a tone to indicate VS1053 is working

if (!SD.begin(CARDCS)) {

Serial.println(F("SD failed, or not present"));

while (1); // don't do anything more

}

Serial.println("SD OK!");

// list files

printDirectory(SD.open("/"), 0);

// Set volume for left, right channels. lower numbers == louder volume!

musicPlayer.setVolume(0,0);

#if defined(\_\_AVR\_ATmega32U4\_\_)

// Timer interrupts are not suggested, better to use DREQ interrupt!

// but we don't have them on the 32u4 feather...

musicPlayer.useInterrupt(VS1053\_FILEPLAYER\_TIMER0\_INT); // timer int

#else

// If DREQ is on an interrupt pin we can do background

// audio playing

musicPlayer.useInterrupt(VS1053\_FILEPLAYER\_PIN\_INT); // DREQ int

#endif

pinMode(buttonPin, INPUT\_PULLUP);

delay( 3000 ); //safety startup delay

FastLED.setMaxPowerInVoltsAndMilliamps( VOLTS, MAX\_MA);

FastLED.addLeds<LED\_TYPE,DATA\_PIN,COLOR\_ORDER>(leds, NUM\_LEDS)

.setCorrection(TypicalLEDStrip);

chooseNextColorPalette(gTargetPalette);

}

void loop()

{

int buttonState;

buttonState = digitalRead(buttonPin);

if (buttonState == LOW) {

delay(500);

Serial.println("color change");

chooseNextColorPalette( gTargetPalette);

}

EVERY\_N\_MILLISECONDS( 1 ) {

nblendPaletteTowardPalette( gCurrentPalette, gTargetPalette, 12);

}

if (counter == 1) {

musicPlayer.startPlayingFile("/track003.mp3");

}

else if (counter == 2) {

musicPlayer.startPlayingFile("/track002.mp3");

}

else if (counter == 3) {

musicPlayer.startPlayingFile("/track004.mp3");

}

else if (counter == 4) {

musicPlayer.startPlayingFile("/track001.mp3");

}

else if (counter == 5) {

musicPlayer.startPlayingFile("/track005.mp3");

}

else if (counter == 6) {

FastLED.clear();

FastLED.show();

}

drawTwinkles(leds);

FastLED.show();

while (musicPlayer.playingMusic) {

buttonState = digitalRead(buttonPin);

if (buttonState == LOW) {

delay(150);

Serial.println("color change");

musicPlayer.stopPlaying();

chooseNextColorPalette( gTargetPalette);

}

EVERY\_N\_MILLISECONDS( 1 ) {

nblendPaletteTowardPalette( gCurrentPalette, gTargetPalette, 12);

}

drawTwinkles(leds);

FastLED.show();

}

}

// This function loops over each pixel, calculates the

// adjusted 'clock' that this pixel should use, and calls

// "CalculateOneTwinkle" on each pixel. It then displays

// either the twinkle color of the background color,

// whichever is brighter.

void drawTwinkles( CRGBSet& L)

{

// "PRNG16" is the pseudorandom number generator

// It MUST be reset to the same starting value each time

// this function is called, so that the sequence of 'random'

// numbers that it generates is (paradoxically) stable.

uint16\_t PRNG16 = 11337;

uint32\_t clock32 = millis();

// Set up the background color, "bg".

// if AUTO\_SELECT\_BACKGROUND\_COLOR == 1, and the first two colors of

// the current palette are identical, then a deeply faded version of

// that color is used for the background color

CRGB bg;

if( (AUTO\_SELECT\_BACKGROUND\_COLOR == 1) &&

(gCurrentPalette[0] == gCurrentPalette[1] )) {

bg = gCurrentPalette[0];

uint8\_t bglight = bg.getAverageLight();

if( bglight > 64) {

bg.nscale8\_video( 16); // very bright, so scale to 1/16th

} else if( bglight > 16) {

bg.nscale8\_video( 64); // not that bright, so scale to 1/4th

} else {

bg.nscale8\_video( 86); // dim, scale to 1/3rd.

}

} else {

bg = gBackgroundColor; // just use the explicitly defined background color

}

uint8\_t backgroundBrightness = bg.getAverageLight();

for( CRGB& pixel: L) {

PRNG16 = (uint16\_t)(PRNG16 \* 2053) + 1384; // next 'random' number

uint16\_t myclockoffset16= PRNG16; // use that number as clock offset

PRNG16 = (uint16\_t)(PRNG16 \* 2053) + 1384; // next 'random' number

// use that number as clock speed adjustment factor (in 8ths, from 8/8ths to 23/8ths)

uint8\_t myspeedmultiplierQ5\_3 = ((((PRNG16 & 0xFF)>>4) + (PRNG16 & 0x0F)) & 0x0F) + 0x08;

uint32\_t myclock30 = (uint32\_t)((clock32 \* myspeedmultiplierQ5\_3) >> 3) + myclockoffset16;

uint8\_t myunique8 = PRNG16 >> 8; // get 'salt' value for this pixel

// We now have the adjusted 'clock' for this pixel, now we call

// the function that computes what color the pixel should be based

// on the "brightness = f( time )" idea.

CRGB c = computeOneTwinkle( myclock30, myunique8);

uint8\_t cbright = c.getAverageLight();

int16\_t deltabright = cbright - backgroundBrightness;

if( deltabright >= 32 || (!bg)) {

// If the new pixel is significantly brighter than the background color,

// use the new color.

pixel = c;

} else if( deltabright > 0 ) {

// If the new pixel is just slightly brighter than the background color,

// mix a blend of the new color and the background color

pixel = blend( bg, c, deltabright \* 8);

} else {

// if the new pixel is not at all brighter than the background color,

// just use the background color.

pixel = bg;

}

}

}

// This function takes a time in pseudo-milliseconds,

// figures out brightness = f( time ), and also hue = f( time )

// The 'low digits' of the millisecond time are used as

// input to the brightness wave function.

// The 'high digits' are used to select a color, so that the color

// does not change over the course of the fade-in, fade-out

// of one cycle of the brightness wave function.

// The 'high digits' are also used to determine whether this pixel

// should light at all during this cycle, based on the TWINKLE\_DENSITY.

CRGB computeOneTwinkle( uint32\_t ms, uint8\_t salt)

{

uint16\_t ticks = ms >> (8-TWINKLE\_SPEED);

uint8\_t fastcycle8 = ticks;

uint16\_t slowcycle16 = (ticks >> 8) + salt;

slowcycle16 += sin8( slowcycle16);

slowcycle16 = (slowcycle16 \* 2053) + 1384;

uint8\_t slowcycle8 = (slowcycle16 & 0xFF) + (slowcycle16 >> 8);

uint8\_t bright = 0;

if( ((slowcycle8 & 0x0E)/2) < TWINKLE\_DENSITY) {

bright = attackDecayWave8( fastcycle8);

}

uint8\_t hue = slowcycle8 - salt;

CRGB c;

if( bright > 0) {

c = ColorFromPalette( gCurrentPalette, hue, bright, NOBLEND);

if( COOL\_LIKE\_INCANDESCENT == 1 ) {

}

} else {

c = CRGB::Black;

}

return c;

}

// This function is like 'triwave8', which produces a

// symmetrical up-and-down triangle sawtooth waveform, except that this

// function produces a triangle wave with a faster attack and a slower decay:

//

// / \

// / \

// / \

// / \

//

uint8\_t attackDecayWave8( uint8\_t i)

{

if( i < 86) {

return i \* 3;

} else {

i -= 86;

return 255 - (i + (i/2));

}

}

// This function takes a pixel, and if its in the 'fading down'

// part of the cycle, it adjusts the color a little bit like the

// way that incandescent bulbs fade toward 'red' as they dim.

void coolLikeIncandescent( CRGB& c, uint8\_t phase)

{

if( phase < 192) return;

uint8\_t cooling = (phase - 192) >> 4;

c.g = qsub8( c.g, cooling);

c.b = qsub8( c.b, cooling \* 2);

}

// A mostly (dark) green palette with red berries.

#define Holly\_Green 0x00580c

const TProgmemRGBPalette16 Green\_p FL\_PROGMEM =

{ Holly\_Green, Holly\_Green, Holly\_Green, Holly\_Green,

Holly\_Green, Holly\_Green, Holly\_Green, Holly\_Green,

Holly\_Green, Holly\_Green, Holly\_Green, Holly\_Green,

Holly\_Green, Holly\_Green, Holly\_Green, Holly\_Green

};

// A yellow pllette.

#define yellow 0xfffb00

const TProgmemRGBPalette16 Yellow\_p FL\_PROGMEM =

{ yellow, yellow, yellow, yellow,

yellow, yellow, yellow, yellow,

yellow, yellow, yellow, yellow,

yellow, yellow, yellow, yellow

};

// A blue pllette.

#define blue 0x2986cc

const TProgmemRGBPalette16 Blue\_p FL\_PROGMEM =

{ blue, blue, blue, blue,

blue, blue, blue, blue,

blue, blue, blue, blue,

blue, blue, blue, blue

};

// A white pllette.

#define white 0xC0C0C0

const TProgmemRGBPalette16 White\_p FL\_PROGMEM =

{ white, white, white, white,

white, white, white, white,

white, white, white, white,

white, white, white, white

};

// A pink pllette.

#define pink 0xff00a5

const TProgmemRGBPalette16 Pink\_p FL\_PROGMEM =

{ pink, pink, pink, pink,

pink, pink, pink, pink,

pink, pink, pink, pink,

pink, pink, pink, pink

};

// A red pllette.

#define red 0xff0000

const TProgmemRGBPalette16 Red\_p FL\_PROGMEM =

{ red, red, red, red,

red, red, red, red,

red, red, red, red,

red, red, red, red

};

// A red pllette.

#define black 0x000000

const TProgmemRGBPalette16 BLK\_p FL\_PROGMEM =

{ black, black, black, black,

black, black, black, black,

black, black, black, black,

black, black, black, black

};

// Add or remove palette names from this list to control which color

// palettes are used, and in what order.

const TProgmemRGBPalette16\* ActivePaletteList[] = {

&White\_p,

&Green\_p,

&Yellow\_p,

&Blue\_p,

&Pink\_p,

&Red\_p,

&BLK\_p,

};

// Advance to the next color palette in the list (above).

void chooseNextColorPalette( CRGBPalette16& pal)

{

const uint8\_t numberOfPalettes = sizeof(ActivePaletteList) / sizeof(ActivePaletteList[0]);

static uint8\_t whichPalette = -1;

whichPalette = addmod8( whichPalette, 1, numberOfPalettes);

pal = \*(ActivePaletteList[whichPalette]);

Serial.println( whichPalette);

counter = whichPalette;

}

/// File listing helper

void printDirectory(File dir, int numTabs) {

while(true) {

File entry = dir.openNextFile();

if (! entry) {

// no more files

//Serial.println("\*\*nomorefiles\*\*");

break;

}

for (uint8\_t i=0; i<numTabs; i++) {

Serial.print('\t');

}

Serial.print(entry.name());

if (entry.isDirectory()) {

Serial.println("/");

printDirectory(entry, numTabs+1);

} else {

// files have sizes, directories do not

Serial.print("\t\t");

Serial.println(entry.size(), DEC);

}

entry.close();

}

}