```
/////  Project: 14 Band Spectrum Analyzer using WS2812B addressable "Smart" LED's and MSGEQ7 band-slicing IC's
/////  Programmed and tested by Daniel Perez, A.K.A GeneratorLabs
/////  Location: Myrtle Beach, South Carolina, United States of America
//N//  E-Mail: generatorlabs@gmail.com
//O//  Date: June 01, 2019
//E//  Revision: Ver 2.3
//T//  Target Platform: Arduino Mega2650 with SpeckyBoard
//E//  License: This program is free software. You can redistribute it and/or modify it under the terms of the GNU General Public License as published by
//S//  the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
/////  This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
/////  FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for more details.
/////  Credits: See acknowledgements & credits section below
/*
  /////  More information about this project, the necessary circuits and a source for parts-kits can be obtained by email (generatorlabs@gmail.com)
  /////  The notes & comments do not consume Arduino memory. They automatically get stripped out before compiled program is uploaded to Arduino.
  /////  Please keep notes in tact for future reference.
  /////
  /////  --- CREDITS & ACKNOWLEDGEMENTS ---
  /////  This sketch is based on a similar project and code from Denis Lanfrit and I respectfully thank Denis for his open source efforts.
  //N//  The original code has been modified to allow a scalable platform with many more bands.
  //O//  The library Si5351mcu is being utilized for programming masterclock IC frequencies. Special thanks to Pavel Milanes for his outstanding work. (https://github.com/pavelmc/Si5351mcu)
  //T//  The library "FastLED" is being utilized to send data to the LED string. This library was developed by Daniel Garcia in 2012 and I respectfully
  //E//  thank him for the amazing work as well. (https://github.com/FastLED/FastLED)
  //S//
  /////  This sketch is written for an Arduino Mega2650. While it is possible to run modified code on a UNO or NANO, the memory limitations of those
  /////  devices makes it impractical for a spectrum analyzer operating with more than 7 bands. The cost difference between the UNO and Mega2560 is so small that
  /////  it makes no sense to cram code into an UNO with only a few bytes to spare.
  /////
  //N//  --- PIN ASSIGNMENTS ---
  //O//
  //T//  Smart LED's use Pin 36 for data. The LED's are defined as a single wire WS2812B type in "SETUP" section.
  //E//  Strobe signal uses pin 7
  //S//  Reset signal uses pin 6
  /////  Analog reading of MSGEQ7 IC1 and IC2 use pin A0 and A1 respectively.
  /////  Make sure all boards, sub-assemblies, LED strings, etc are all tied to a common ground. Failure to do so will result in erratic operation and
  /////  possible circuit or component failure. Treat smart LED's with respect. They are susceptable to electrostatic discharge and improper voltages & polarity!
  /////  This code is being offered AS-IS. End user assumes any responsibility associated with the use of this code.
*/


#include <si5351mcu.h>    // Library used to program clock generator IC via I2C
Si5351mcu Si;             // Library instantiation as "Si"

#include <FastLED.h>      // You must include FastLED version 3.002.006. This library allows communication with each LED

#define HEARTBEAT_PIN 5   // Pin for heartbeat
#define DATA_PIN 36       // Pin for serial communication with LED string. This pin directs data thru termination resistor R13 on my 'SPECKY-BOARD'.
#define STROBE_PIN 7      // Pin to instruct MSGEQ7 IC's to inspect next band (band 0 thru 6). Default Pin is 6. Default Pin on SpeckyBoard is 7.
#define RESET_PIN 6       // Pin to instruct MSGEQ7 to return to band zero and inspect it. Default Pin is 7. Default Pin on SpeckyBoard is 6.

#define NUM_LEDS 294      // Total number of LED's in the project. Should be equal to COLUMN * ROWS
#define COLUMN 14         // Number of columns in LED project
#define ROWS 21           // Number of rows (left to right) in LED project.
#define BRIGHTNESS  50    // Intensity of LED's. The lower the number the longer LED's will last. LED's do have a finite life span when run at high output.
// It is strongly recommended to keep this number as low as possible. Inexpensive LED strips will have a noticeably shorter life and pull large
// amounts of current unecessarily. Large surges in current could lead to current starvation and possibly erratic operation. Your power supply must be
// sized correctly. A string of 300 LED's could potentially require a 18 amp power supply! Avoid drawing white as a color if your power supply is substandard
// or poorly regulated. For reference, my 294 LED analyzer, with a brightness of 50 and static rainbow columns will average less than
// 0.5 amps @ 5vdc. If you drive the LED's conseratively you will get good results.

// Noise-floor compensator. Set this number to eliminate noise picked up by circuit. When watching serial monitor, data
// should be closer to zero with an audio source connected and no music playing.
#define NOISECOMP 200     // 65 is a good start point. ; My number is 140

// Use a frequency generator app on a smart phone to adjust Delta. All led's should in selected band should light up when outputting center frequencies at high volume.
// Test on 63Hz, 160Hz, 400Hz, & 1000Hz. Clock signals being fed into MSGEQ IC's must be accurately tuned or band drifting will occur and cause adjustments to be skewed.
#define DELTA 38          // 48 is a good start point. ; My number 42

#define LEDTYPE WS2812B   // Type of LED communication protocol used.
```

```
// Matrix Definition
CRGB leds[NUM_LEDS];
typedef struct ledrgb     // Structure defining the parameters related to each led
{
  int r;
  int g;
  int b;
  int hue;
  int sat;
  int val;
  int nled;
  boolean active;
} led;
led colors[COLUMN][ROWS];   // Matrix containing the progressive number of each single LED

//Global Variables
int MSGEQ_Bands[COLUMN];    // Setup column array
int nlevel;                 // Level index
int hue_rainbow = 0;        // Global variable for the rainbow variable hue
int long rainbow_time = 0;
int long time_change = 0;
int long heartbeat = 0;
int effect = 2;             // Load this color effect on startup
bool toggle = false;
int n = 0;

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////------------------- SETUP ----------------------/////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void setup()
{
  // Serial.begin(57600);      // Enable this for serial monitor or troubleshooting

// Start Masterclock configuration; The remainder of this program will fail if this does not initialize!
Si.init(25000000L);          // Library procedure to set up for use with non-default 25.000 MHz xtal
Si.setFreq(0, 165000);       // Enable the output 0  with specified frequency of 165.000 KHz; Default Pin for SpeckyBoard
Si.setFreq(1, 104000);       // Enable the output 1  with specified frequency of 104.000 KHz; Default Pin for SpeckyBoard
Si.setPower(0, SIOUT_8mA);   // Set power output level of clock 0
Si.setPower(1, SIOUT_8mA);   // Set power output level of clock 1
Si.enable(0);                // Enable output 0
Si.enable(1);                // Enable output 1
// End Masterclock configuration


pinMode(HEARTBEAT_PIN, OUTPUT);
pinMode(DATA_PIN, OUTPUT);
pinMode(STROBE_PIN, OUTPUT);
pinMode(RESET_PIN, OUTPUT);
digitalWrite(HEARTBEAT_PIN, HIGH);    // OK! Si5351 clock passed initializiation!
delay(100);
digitalWrite(HEARTBEAT_PIN, LOW);
delay(100);
digitalWrite(HEARTBEAT_PIN, HIGH);
delay(100);
digitalWrite(HEARTBEAT_PIN, LOW);
delay(100);
digitalWrite(HEARTBEAT_PIN, HIGH);
delay(100);

int n = 0;
for (int i = 0; i < COLUMN; i++)      //Sequentially number the leds
{
  for (int j = 0; j < ROWS; j++)
  {
    colors[i][j].nled = n;
```

```
      n++;
    }
  }

  FastLED.addLeds<LEDTYPE, DATA_PIN, GRB>(leds, NUM_LEDS).setCorrection( TypicalLEDStrip );
  FastLED.setBrightness( BRIGHTNESS );
  rainbow_time = millis();
  time_change = millis();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////-------------------- LOOP ----------------------///////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void loop()
{
  readMSGEQ7();                                 // Call to function that reads MSGEQ7 IC's via analogue inputs.

  if (millis() - time_change > 1000)            // Code that establishes how often to change effect. 1000 = 1 Second
  {
    effect = 2;                                 // Enable this line to set a fixed mode
    //effect++;                                 // Enable this line to cycle through different modes
    if (effect > 7) {
      effect = 0;
    }
    time_change = millis();
  }

  if (millis() - heartbeat > 3000)              // Hearbeat LED to indicate that code has passed init and is actually looping through routines
  {
    toggle = !toggle;
    digitalWrite(HEARTBEAT_PIN, toggle);
    heartbeat = millis();
  }

switch (effect)                                 // Case logic to determine which color effect to use
{
  case 0:                                       // Full column; each band different color; color gradient within each band
    rainbow_dot();
    full_column();
    updateHSV();
    break;

  case 1:                                       // Full column; each band the same color; gradual simultaneous color change across all bands
    if (millis() - rainbow_time > 15)
    {
      dynamic_rainbow();
      rainbow_time = millis();
    }
    full_column();
    updateHSV();
    break;

  case 2:                                       // Full column; each band a different static rainbow color for the specified interval
    if (millis() - rainbow_time > 600)
    {
      rainbow_column();
      rainbow_time = millis();
    }
    full_column();
    updateHSV();
    break;

  case 3:                                       // Full column; all bands same static color
    if (millis() - rainbow_time > 15)
    {
      total_color_hsv(255, 255, 255);
      rainbow_time = millis();
```

```
    }
    full_column();
    updateHSV();
    break;

  case 4:                                 // Dot column; each column a different static rainbow color
    if (millis() - rainbow_time > 15)
    {
      rainbow_dot();
      rainbow_time = millis();
    }
    full_column_dot();;
    updateHSV();
    break;
  case 5:                                 // Dot column; each band the same color; gradual simultaneous color change across all bands
    if (millis() - rainbow_time > 15)
    {
      dynamic_rainbow();
      rainbow_time = millis();
    }
    full_column_dot();
    updateHSV();
    break;

  case 6:                                 // Dot column; each band a different static rainbow color
    if (millis() - rainbow_time > 15)
    {
      rainbow_column();
      rainbow_time = millis();
    }
    full_column_dot();
    updateHSV();
    break;

  case 7:                                 // Dot column; all bands same static color
      total_color_hsv(55, 255, 255);
      full_column_dot();
      updateHSV();
      break;
  }
  delay(30);                              // Refresh rate; Values 20 thru 30 should look realistic
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////----------------- FUNCTIONS ---------------------/////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void readMSGEQ7(void)                     // Function that reads the 7 bands of the audio input.
{
  //digitalWrite(STROBE_PIN, HIGH);       // Make sure Strobe line is low before entering loop.

  digitalWrite(RESET_PIN, HIGH);          // Part 1 of Reset Pulse. Reset pulse duration must be 100nS minimum.
  digitalWrite(RESET_PIN, LOW);           // Part 2 of Reset pulse. These two events consume more than 100nS in CPU time.

    for (int band = 0; band < COLUMN; band++) {     // Loop that will increment counter that AnalogRead uses to determine which band to store data for.
    digitalWrite(STROBE_PIN, LOW);        // Re-Set Strobe to LOW on each iteration of loop.
    delayMicroseconds(30);                // Necessary delay required by MSGEQ7 for proper timing.
    MSGEQ_Bands[band] = analogRead(0) - NOISECOMP;  // Saves the reading of the amplitude voltage on Analog Pin 0.
        // Serial.print(band);
        // Serial.print("  ");
        // Serial.print(MSGEQ_Bands[band]);
        // Serial.print("  ");
    band++;
    MSGEQ_Bands[band] = analogRead(1) - NOISECOMP;  // Saves the reading of the amplitude voltage on Analog Pin 1.
        // Serial.print(band);
        // Serial.print("  ");
        // Serial.print(MSGEQ_Bands[band]);
        // Serial.print("  ");
```

```
    digitalWrite(STROBE_PIN, HIGH);
  }
        // Serial.println();
        //digitalWrite(STROBE_PIN, LOW);            // Make sure Strobe line is low before entering loop.
}


void updateRGB(void)
{
  for (int i = 0; i < COLUMN; i++) {
    for (int j = 0; j < ROWS; j++) {
      if (colors [i][j].active == 1) {
        leds[colors[i][j].nled] = CRGB(colors[i][j].r, colors[i][j].g, colors[i][j].b);
      } else {
        leds[colors[i][j].nled] = CRGB::Black;
      }
      FastLED.show();
    }
  }
}

void updateHSV(void)
{
  for (int i = 0; i < COLUMN; i++) {
    for (int j = 0; j < ROWS; j++) {
      if (colors[i][j].active == 1) {
        leds[colors[i][j].nled] = CHSV(colors[i][j].hue, colors[i][j].sat, colors[i][j].val);
      } else {
        leds[colors[i][j].nled] = CRGB::Black;
      }

    }
  }
  FastLED.show();
}

void full_column(void)
{
  nlevel = 0;
  for (int i = 0; i < COLUMN; i++) {
    nlevel = MSGEQ_Bands[i] / DELTA;
    for (int j = 0; j < ROWS; j++) {
      if (j <= nlevel) {
        colors[i][j].active = 1;
      }
      else {
        colors[i][j].active = 0;
      }
    }
    //colors[i][peaks[i]].active=1;
  }
}

void full_column_dot(void)
{
  nlevel = 0;
  for (int i = 0; i < COLUMN; i++) {
    nlevel = MSGEQ_Bands[i] / DELTA;
    for (int j = 0; j < ROWS; j++) {
      if (j == nlevel) {
        colors[i][j].active = 1;
      }
      else {
        colors[i][j].active = 0;
      }
    }
    //colors[i][peaks[i]].active=1;
  }
```

```
}

void total_color_hsv(int h, int s, int v)
{
  for (int i = 0; i < COLUMN; i++) {
    for (int j = 0; j < ROWS; j++) {
      colors[i][j].hue = h;
      colors[i][j].sat = s;
      colors[i][j].val = v;
    }
  }
}

void total_color_rgb(int r, int g, int b)
{
  for (int i = 0; i < COLUMN; i++) {
    for (int j = 0; j < ROWS; j++) {
      colors[i][j].r = r;
      colors[i][j].g = g;
      colors[i][j].b = b;
    }
  }
}

void rainbow_column(void)
{
  //int n = 18;
  for (int i = 0; i < COLUMN; i++) {
    for (int j = 0; j < ROWS; j++) {
      colors[i][j].hue = n;
      colors[i][j].sat = 230;
      colors[i][j].val = 240;
    }
    n += 18;  //36 For 7 Columns
  }
}

void rainbow_dot(void)
{
  int n = 36;
  for (int i = 0; i < COLUMN; i++) {
    for (int j = 0; j < ROWS; j++) {
      colors[i][j].hue = n;
      colors[i][j].sat = 230;
      colors[i][j].val = 240;
      n += 5;
    }
  }
}

void dynamic_rainbow(void)
{
  for (int i = 0; i < COLUMN; i++) {
    for (int j = 0; j < ROWS; j++) {
      colors[i][j].hue = hue_rainbow;
      colors[i][j].sat = 230;
      colors[i][j].val = 240;
    }
  }
  hue_rainbow++;
}
```