An Electronic Christmas Tree

Contents

Description

Figure 1 Front View of the Finished Christmas Tree

This document describes the hardware and software details of a Do-It-Yourself Electronic Christmas Tree. This project incorporates 14 WS2812 multi-color addressable LED's which are driven by a ATMega328 microcontroller. The software is written in C++ on an Arduino boot load and can easily be changed.

The Tree displays pseudo-random colors in all the attached LED's and is powered by a standard 9 Volt battery, which is inserted into a 3D printed holder as in Figure 2. The Tree will automatically shut off after about 2 hours in order to conserve the battery. Note that even while shut off, the microcontroller will continue to draw a small amount of current and you should disconnect the battery if you are not planning on showing the display for a while.

After shutting off automatically, the display can resume by pressing the pushbutton switch located in the middle of the front of the tree.

Figure 1 shows the finished product, an electronic Christmas Tree.

Figure 2 Rear Side of Christmas Tree

Hardware Details

The core of this design is an Atmel ATMega328 processor U1 and associated circuitry, Y1, R3, R2, C2, C3, C4, and C5. You will quickly recognize this as a stripped-down Arduino. This allows you to program the code on an Arduino UNO and then move the processor to this project. Alternatively, you can just use an Arduino UNO and wire up the additional circuitry to it.

The ATMega328 sends serial digital commands to the WS2812B's connected to port PD6 (Arduino pin 6) to address each individually and change the color being displayed. Note the use of WS2812B. These are used since they are 5 Volt powered, instead of needing 12 Volts like the WS2811 and are more robust than the WS2812 version.

The U2, C1 combination converts the battery's 9V power to the 5V required by the rest of the circuitry.

Note, the J1 USB port is not used in Version A of the PCB due to a problem with the pad sizes being used which caused a short between +5V and Ground. You will note on Figure 1 that a cut has been made between pin 5 (right most pin) and the surrounding PCB plane, which is connected to +5VDC. A fix was made in Version B of the PCB to remove the connector, thus eliminating the problem. As a result, the additional free space on the PCB was given to an addition 2 LED's. The project files referenced in the appendices all relate to Version A, however, since Version B has not yet been manufactured and tested.

Also on Version A, you will note in Figure 2 that a barnacle (mod wire) has been added to the switch. This connects the pushbutton switch to Port PD2 (Arduino pin 2) as well as Port PD7 (Arduino pin 7), which is as shown on the schematic. This change permits the pushbutton switch to "wake up" the processor after it has been shut down after the 2-hour timeout. Port D2 is pin 4 of the ATMega328 processor.

If you want build your Christmas Tree using the Version A PCB, I have included a link to the Gerber Zip file in Appendix 3. Note: your browser may balk at downloading a Zip file. If this concerns you, contact me.

You can order PCB's using this file from just about any PCB house. I use JLPCB in China and OshPark in the USA for PCB's.

Software Details

I would like to acknowledge Mark Kriegsman for his work in developing the LED display code that I used for this project. That library (FastLED) is included in the project's code.

I would also like to acknowledge "The Arduino Man" for his guidance on using interrupts on the ATMega328.

Please reference Appendix 4 for the code details.

Like all Arduino software, the code begins by setting up global parameters and initializes the FastLED library with settings for the pin (6) used to communicate with the LEDs, the type of LEDs (WS2812B) and the quantity being used (14). Brightness of the LEDs (96 out of a possible 255) and Frames Per Second are also established at this point. We also define how long to display (120 minutes = 2 hours) and establish pin 2 as the interrupt source.

After this, the "Setup" function sets up the interrupt, the serial port if being used for Debug (disabled in the attached code), and communicates the parameters to the FastLED library. The "patterns" of the LED's is then setup via the SimplePatternList. These can be changed at will.

The next function is the Main Loop. If Debug is enabled, the pattern being displayed will be output to a connected terminal. The loaded pattern is displayed via the "FastLED.show" function. Every 20 milli-seconds, the color of the LED's will be changed. Every 10 Seconds, the pattern will get changed to the

next in line as per the SImplePatterns list. Lastly, the loop will check to see if 2 hours have elapsed, and if so, will shut down the processor.

There are a number of functions following the Main Loop. These are used within the Main Loop or elsewhere and should be self-explanatory. The function “GoingToSleep” is important – it will do the processor shutdown after establishing which pin will wake it back up.

Conclusion

I hope that this project has given you some ideas to work on a similar project of your own.

73

Mitch NØDIM

Appendix 1 – Schematic

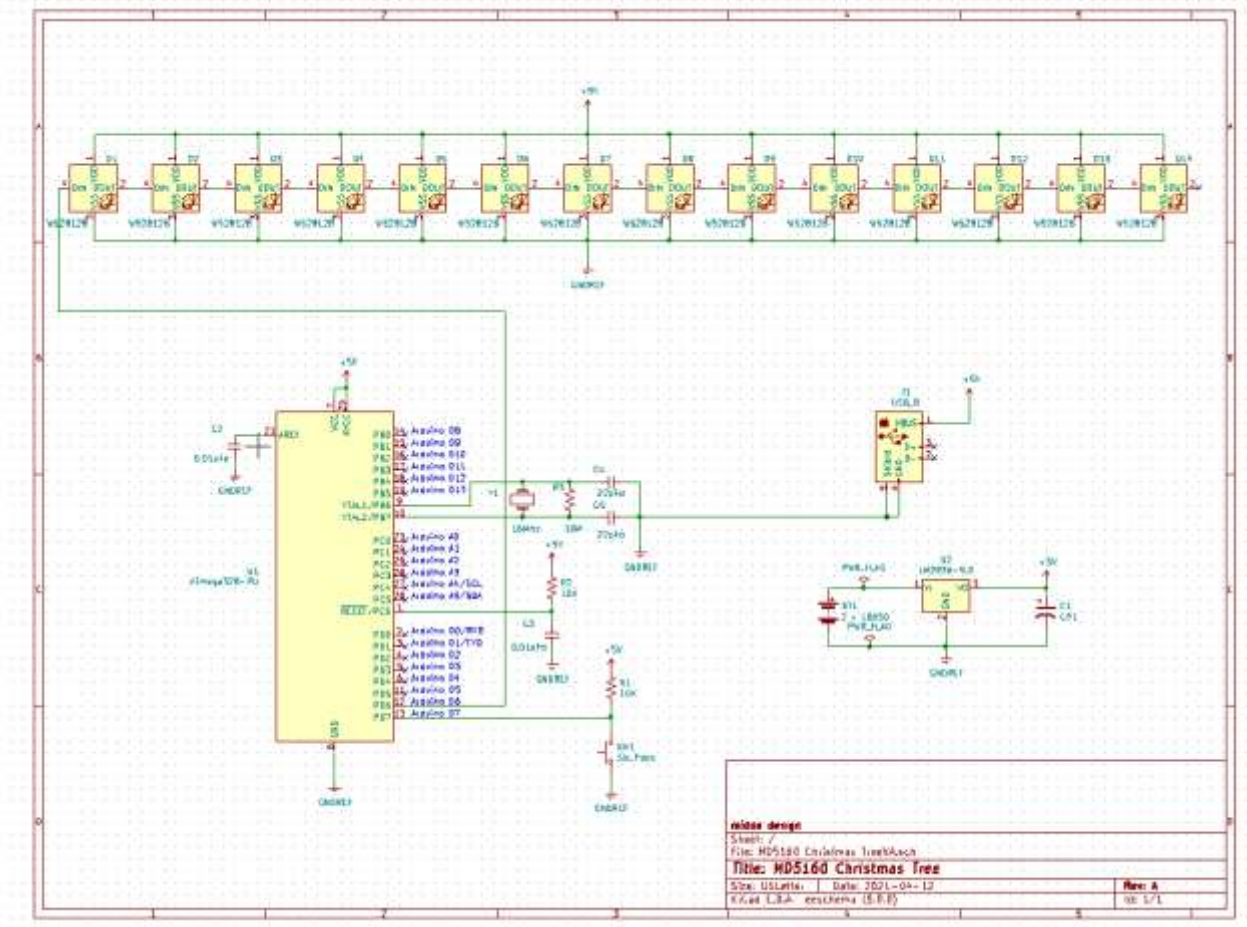

Figure 3 MD5160 Electronic Christmas Tree Schematic

Appendix 2 – Bill of Material

Designator	Quantity	Designation
C4,C5	2	22pFd disk capacitor
C1	1	10uFd 16V electrolytic capacitor
C2, C3	2	0.01uFd disk capacitor
R3	1	10M ¼ W resistor
R1	1	10K ¼ W resistor
U1	1	ATmega328P-PU
U2	1	LM78L05
D1-D14	14	WS2812B
Y1	1	16MHz

Appendix 3 - Sources

1. WS2812B LED: https://cdn-shop.adafruit.com/datasheets/WS2812B.pdf
2. ATMega328 Processor with Arduino Bootloader: https://www.jameco.com/z/A000048-Arduino-ATMega328P-32kB-Microcontroller-with-Arduino-Uno-Optiboot-Bootloader_2129334.html
3. PCB Gerbers for this project: http://n0dim.com/Documents/MD5160 Christmas Tree VA.zip
4. PCB manufacturers: https://jlcpcb.com/ and https://oshpark.com/
5. FastLED library: http://fastled.io/
6. STL file for Christmas Tree base: http://n0dim.com/Documents/MD5160 Christmas Tree Stand 2021.stl

Appendix 4 – Arduino Code

```
#include <FastLED.h>
#include <avr/sleep.h>

FASTLED_USING_NAMESPACE

//-------------------------------------------------------------------------------------
// FastLED "100-lines-of-code" demo reel, showing just a few
// of the kinds of animation patterns you can quickly and easily
// compose using FastLED.
//
// This example also shows one easy way to define multiple
// animations patterns and have them automatically rotate.
//
// -Mark Kriegsman, December 2014
//
//
//  Sleep Interrupt code taken from https://thekurks.net/blog/2018/1/24/guide-to-
arduino-sleep-mode
//
//-------------------------------------------------------------------------------------


//#if defined(FASTLED_VERSION) && (FASTLED_VERSION < 3001000)
//#warning "Requires FastLED 3.1 or later; check github for latest code."
//#endif
//-------------------------------------------------------------------------------------
#define DEVMODE 0         // set to ZERO if not debugging or ONE if you want to.
115200 BAUD Rate required.
//-------------------------------------------------------------------------------------

#define DATA_PIN    6
//#define CLK_PIN   4
#define interruptPin 2    // MUST be either pin D2 or D3!  Barnacle required on VA PCB
to connect pins 2 and 7.
#define LED_TYPE    WS2812B
#define COLOR_ORDER GRB
#define NUM_LEDS    14
CRGB leds[NUM_LEDS];

#define BRIGHTNESS          96
#define FRAMES_PER_SECOND  120

#define AwakeTime 120     // number of minutes to display

int TIMER_MINUTES = 0;
int TIMER_SECONDS = 0;

//-------------------------------------------------------------------------------------
//            S E T U P
//-------------------------------------------------------------------------------------
void setup() {
  pinMode(interruptPin,INPUT);
  pinMode(7,INPUT);           // needed for VA of the PCB
  delay(3000);                // 3 second delay for recovery
if (DEVMODE) {
  Serial.begin(115200);
  Serial.println("MD5160 Christmas Tree 210510");
}
  // tell FastLED about the LED strip configuration
```

```
  FastLED.addLeds<LED_TYPE,DATA_PIN,COLOR_ORDER>(leds,
NUM_LEDS).setCorrection(TypicalLEDStrip);
  //FastLED.addLeds<LED_TYPE,DATA_PIN,CLK_PIN,COLOR_ORDER>(leds,
NUM_LEDS).setCorrection(TypicalLEDStrip);

  // set master brightness control
  FastLED.setBrightness(BRIGHTNESS);
}


// List of patterns to cycle through.  Each is defined as a separate function below.
typedef void (*SimplePatternList[])();
SimplePatternList gPatterns = { rainbow, rainbowWithGlitter, confetti, sinelon,
juggle, bpm };

uint8_t gCurrentPatternNumber = 0; // Index number of which pattern is current
uint8_t gHue = 0; // rotating "base color" used by many of the patterns

//---------------------------------------------------------------------------------
//            M A I N   L O O P
//---------------------------------------------------------------------------------
void loop()
{
  // Call the current pattern function once, updating the 'leds' array
  gPatterns[gCurrentPatternNumber]();
  if (DEVMODE) {
  Serial.print("Current Pattern: ");
  Serial.println(gCurrentPatternNumber);
  Serial.print("Current Timer Value: ");
  Serial.println(TIMER_SECONDS);
  }
  // send the 'leds' array out to the actual LED strip
  FastLED.show();
  // insert a delay to keep the framerate modest
  FastLED.delay(1000/FRAMES_PER_SECOND);

  // do some periodic updates
  EVERY_N_MILLISECONDS( 20 ) { gHue++; } // slowly cycle the "base color" through the
rainbow
  EVERY_N_SECONDS( 10 ) { nextPattern(); } // change patterns periodically

  if (TIMER_MINUTES>AwakeTime) {
    FastLED.clear(true);

    FastLED.show();
    GoingToSleep();         //after "AwakeTime" minutes, turn off the CPU
    // CPU WILL RETURN HERE ON BUTTON PUSH
    TIMER_MINUTES = 0;      // reset the timer upon wakeup
    TIMER_SECONDS - 0;
  }
}

#define ARRAY_SIZE(A) (sizeof(A) / sizeof((A)[0]))

//---------------------------------------------------------------------------------
//  Add one to the current pattern number, and wrap around at the end
//---------------------------------------------------------------------------------
void nextPattern()
{
  gCurrentPatternNumber = (gCurrentPatternNumber + 1) % ARRAY_SIZE( gPatterns);
  TIMER_SECONDS++;
  if (TIMER_SECONDS==6) {     // reset seconds counter if needed
    TIMER_MINUTES++;          // and up the minutes counter
```

```
    TIMER_SECONDS=0;
  }
}

//------------------------------------------------------------------------------------
//  FastLED's built-in rainbow generator
//------------------------------------------------------------------------------------
void rainbow()
{
  fill_rainbow( leds, NUM_LEDS, gHue, 7);
}

//------------------------------------------------------------------------------------
//  Built-in FastLED rainbow, plus some random sparkly glitter
//------------------------------------------------------------------------------------
void rainbowWithGlitter()
{
  rainbow();
  addGlitter(80);
}

//------------------------------------------------------------------------------------
//  Add Glitter to a pattern
//------------------------------------------------------------------------------------
void addGlitter( fract8 chanceOfGlitter)
{
  if( random8() < chanceOfGlitter) {
    leds[ random16(NUM_LEDS) ] += CRGB::White;
  }
}

//------------------------------------------------------------------------------------
//  Random colored speckles that blink in and fade smoothly
//------------------------------------------------------------------------------------
void confetti()
{
  fadeToBlackBy( leds, NUM_LEDS, 10);
  int pos = random16(NUM_LEDS);
  leds[pos] += CHSV( gHue + random8(64), 200, 255);
}

//------------------------------------------------------------------------------------
//  A colored dot sweeping back and forth, with fading trails
//------------------------------------------------------------------------------------
void sinelon()
{
  fadeToBlackBy( leds, NUM_LEDS, 20);
  int pos = beatsin16( 13, 0, NUM_LEDS-1 );
  leds[pos] += CHSV( gHue, 255, 192);
}

//------------------------------------------------------------------------------------
//  Coloured stripes pulsing at a defined Beats-Per-Minute (BPM)
//------------------------------------------------------------------------------------
void bpm()
{
  uint8_t BeatsPerMinute = 62;
  CRGBPalette16 palette = PartyColors_p;
  uint8_t beat = beatsin8( BeatsPerMinute, 64, 255);
  for( int i = 0; i < NUM_LEDS; i++) { //9948
    leds[i] = ColorFromPalette(palette, gHue+(i*2), beat-gHue+(i*10));
  }
}
```

```
//------------------------------------------------------------------------------------
//  Eight colored dots, weaving in and out of sync with each other
//------------------------------------------------------------------------------------

void juggle() {
  fadeToBlackBy( leds, NUM_LEDS, 20);
  byte dothue = 0;
  for( int i = 0; i < 8; i++) {
    leds[beatsin16( i+7, 0, NUM_LEDS-1 )] |= CHSV(dothue, 200, 255);
    dothue += 32;
  }
}

//------------------------------------------------------------------------------------
//  Sleep the CPU until button pushed
//------------------------------------------------------------------------------------
void GoingToSleep() {

if (DEVMODE) {
  Serial.println("GOING TO SLEEP NOW!");
}
  sleep_enable();       //Enable Sleep Mode
  attachInterrupt(0, wakeUp, LOW);  // attach interrupt to pin D2
  set_sleep_mode(SLEEP_MODE_PWR_DOWN);  // go to FULL sleep
  sleep_cpu();  // activate it!
  // CPU will return here after button push
}
//------------------------------------------------------------------------------------
//  Wake up the CPU after button push
//------------------------------------------------------------------------------------
void wakeUp() {
  sleep_disable();      // turn off sleep mode
  detachInterrupt(0);   //and remove the interrupt from pin 7
}
```

END OF CODE