**Transmit the Twilight Zone Theme Once Per Second to AM Radio Using ATTiny85**

Perhaps the world’s most irritating one second timer.

Twilight Zone is trademarked by CBS Broadcasting Inc.

Used here under fair use / educational



This project uses an ATTiny85 processor and the Arduino IDE to transmit a tune in the AM broadcast band. The board used here is the Digispark 16.5MHz model with an onboard voltage regulator and USB interface. You can find it on Amazon for $20USD (less than $3USD if you shop around). The onboard timers on the ATTiny85 are set up to toggle pin PB0 at ~1MHz to create an RF carrier. The RF carrier oscillates at audio frequencies corresponding to four notes in the Twilight Zone theme. The tune repeats exactly once per second until the processor powers off. For this build, you need a microcontroller, a short length of wire, a soldering iron and a suitable battery if you want to go portable.



The Arduino software is short. You can just cut/paste it into the Arduino IDE:

**#include <util/delay.h>**

**// Twilight Zone one second playback at 1MHz RF**

**// For 16.5MHz Digispark ATTINY85**

**// By Craig Hyatt Jul 2025**

**//**

**// Twilight Zone is trademarked by CBS Broadcasting Inc.**

**// Used here under fair use / educational**

**//**

**// CARRIER\_FREQ determines the RF carrier frequency**

**// CARRIER\_FREQ = F\_CPU / (2 \* PBO\_FREQ)**

**// Suppose F\_CPU = 16.5MHz for your processor, then**

**// F\_CPU = 16500000, PBO\_FREQ = 1000000, and so**

**// CARRIER\_FREQ = 16500000 / (2 \* 1000000) -> 8.25**

**// Bumping CARRIER\_FREQ down to 7 gives 1.025MHz**

**// Or you can tweak OSCAL to tune the system clock**

**///////////////////////////////////////**

**const byte CARRIER\_FREQ = 7;**

**///////////////////////////////////////**

**// set up the hardware, then play the tune once per second**

**const int BLINKER = 1;**

**void setup() {**

 **DDRB |= bit (PB0) ; // Pin 0 is RF out**

 **TCCR0A = TCCR0B = TCNT0 = 0; // Stop timer 0 while setting it up**

 **TCCR0A |= bit (COM0A0); // Toggle OCOA/OCOB on match, non-PWM**

 **TCCR0B |= bit (CS00); // No prescaler for max frequency**

 **TCCR0A |= bit (WGM01); // Start timer in compare mode**

 **OCR0A = CARRIER\_FREQ; // Timer match value determines RF freq**

 **pinMode(BLINKER, OUTPUT); // Set up the LED blinker**

**}**

**// Modulates the RF carrier by turning it off and**

**// on at the frequency of each note**

**// G# A G# E is the Twilight Zone theme**

**// G# = 415.3 Hz, A = 440 Hz, E = 329.63 Hz, etc.**

**void loop() {**

 **const float tune [] = {415.3, 440.0, 415.3, 329.63};**

 **const int beat = 250; // note length**

 **for (int i = 0; i < sizeof(tune) / sizeof(float); i++) {**

 **// half period is 0.5 \* 1E6 / f microseconds**

 **float usec = 500000.0 / tune [i];**

 **// snapshot the millisec clock**

 **unsigned long tim = millis();**

 **do { // plays the note for "beat" millisec**

 **TCCR0A &= ~(bit (WGM01)) ;**

 **delayMicroseconds (usec); // delay w/ clock off**

 **TCCR0A |= bit (WGM01) ;**

 **delayMicroseconds (usec); // delay w/ clock on**

 **} while (millis() - tim < beat); // until "beat" ms have passed**

 **}**

 **digitalWrite(BLINKER, HIGH);**

 **delayMicroseconds(1000); // use this to trim the 1Hz period**

 **digitalWrite(BLINKER, LOW);**

**}**

Set up the Arduino IDE for whatever board you want to use. Instructions for setting up the board manager, installing drivers, selecting your processor and so forth will be available for your processor. Here are instructions for the Digispark board: <https://www.youtube.com/watch?v=JGf6P52LO5c> Ultimately, you’ll compile your code and upload it to the processor board:



The board flashes at 1Hz. Tune an AM radio to 1030 and bring it near the board. Tune slowly back and forth until you hear the four notes. You can increase range by soldering a few inches of wire to the RF output pin.

Don’t use more than a few inches of wire, unless you want men in black knocking down your door.

Here’s my processor board with a short wire soldered to PB0 and powered by a 9VDC battery:



In my case, the radio picked up the signal at about 1030 on the AM dial:



You can confirm the frequency using an oscilloscope or frequency counter. My scope and frequency counter measured the carrier frequency to be about 1.030MHz:



Since the output is a square wave, there will be odd harmonics dropping off in power as the reciprocal of the harmonic number. If the fundamental is 1F, output is 1F, 3F, 5F, 7F, 9F, (2*i*-1)F and corresponding power 1, 1/3, 1/5, 1/7, 1/9, 1/(2*i*-1). Additional noise comes from modulating the RF carrier.



Because the transmitter is running at 5VDC and 100% modulation, it will overpower any nearby receiver. I was able to hear the signal on shortwave bands in the 5MHz to 30MHz range and even on broadcast FM from 88MHz to 108MHz─probably from slope detection.

Again, let me caution you not to: connect a long wire to the transmitter, run the transmitter at more than 5VDC nor amplify it, use the transmitter around radios and other equipment that might be damaged by high RF energy nor operate it for long periods of time. Square waves generate “hash” that can interfere with electronics.

Consider this project a jumping off point. If you want to experiment further:

* See if you can make the transmitter work with a standalone ATTiny85 chip running at 16MHz (ATTiny85 needs ~5VDC to run 16MHz internal clock). You can also experiment with porting to other microcontrollers. Port the software to assembly language and switch to integer math. How simple and compact can you make the hardware and software?
* Can you do it with a combination of 555 oscillators? How would you use discrete chips to play the 4 tones in sequence?
* Make an FM transmitter by toggling the frequency instead of switching the RF carrier. For example, you can try modifying OCR0A on the fly. For broadcast FM, the maximum frequency deviation is ±75 kHz to remain within the guard band.
* Sample an arbitrary signal from an analog port on the processor and PWM the RF carrier to broadcast voice and music.
* Instead of playing audio, key the carrier and make a simple CW transmitter. You can transmit Morse code that you can hear on a receiver with a BFO. Be sure to debounce the key switch.
* Figure out how to save power by sleeping the processor and using interrupts instead of running a loop.
* Figure out how to use a 1Hz timer interrupt to trigger the tune instead of running a loop.
* Experiment with the RF generation and modulation scheme to reduce switching noise by switching only at zero crossings and minimizing rapid transitions.
* Experiment with methods to minimize EMI. Add a simple RC low pass filter to the antenna to reduce EMI. Digikey has a calculator at <https://www.digikey.com/en/resources/conversion-calculators/conversion-calculator-low-pass-and-high-pass-filter>. Wind a few turns of the antenna lead through a ferrite bead. Shield the radio in a metal enclosure.
* Make a crystal radio to receive the signal and look at the tuned and demodulated signal with your scope. This shows how AM radio works.