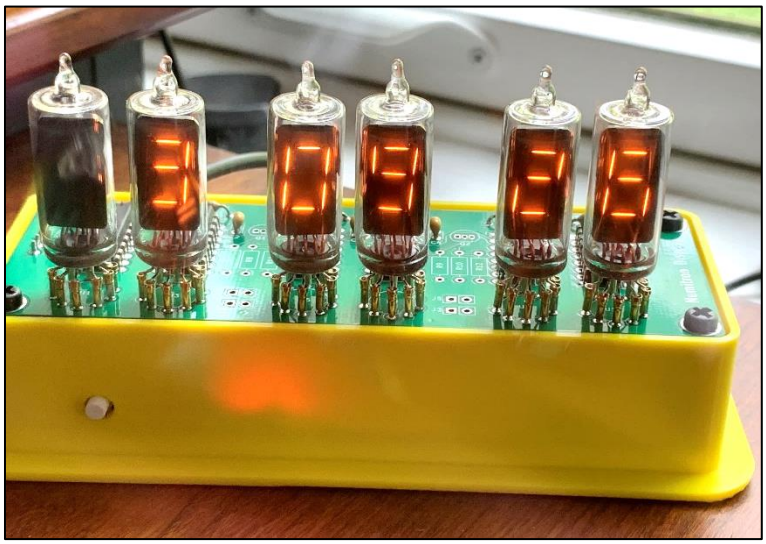
My Numitron Clock

Bruce E. Hall, W8BH



Many articles on this website start off "Build your own [ ]", because they are projects that you might want to build. This article is different. Not many will want to build a Numitron clock. Numitrons haven't been made for many years. And the ones available on eBay are not inexpensive. I document my experience here for those of you who might happen to have a few of these interesting devices laying around…

Introduction

A friend of mine told me that he had a box of original RCA Numitrons. He offered to give me some if I could put them to good use. Without a second thought I said "Of course!" And within a week, I had a small collection of Numitrons on my workbench. Another project was born.

So, what is a Numitron? It is a seven-segment display that uses incandescent filaments in an evacuated tube. They were invented in 1970 by RCA as an alternative to Burroughs Corporation's "Nixie" tube. At that time, LEDs were just starting to enter the market, and were too small and too dim to compete. Read this 1970 Popular Electronics article for a description of the brand-new Numitron device. (Another copy of the same article is here.)

RCA made several different varieties. Mine is the DR2100V1. This tube is their low-power version. At 35mW/segment, it uses 70% less power than its siblings. It also has formed leads, suitable for socketing, and a mean lifetime expectancy of 100,000 hours. Not bad for a fancy lightbulb!

Read the Numitron datasheet for more information.

Driving a Numitron

What is the proper way to drive a Numitron tube? The Popular Electronics article, referenced above, suggested the "CD 2501E" BCD-to-Decimal Decoder, which RCA developed to accompany their Numitron. RCA published a very useful application note which describes how to use Numitrons with this decoder/driver. The CD 2501E is long obsolete, of course.

My first instinct was to drive these displays like an LED. A typical LED display driver is the MAX7221, which interfaces 8 seven-segment displays to a microcontroller using only 3 control lines. It accomplishes this magical feat by multiplexing. The seven-segment displays are driven sequentially at a rapid rate. Persistence of vision makes the display seem completely illuminated.

The MAX7221 turned out to be a poor choice indeed. I forgot that, unlike LEDs, incandescent filaments take time to illuminate. LEDs reach full brightness very quickly; Numitrons do not. Multiplexed Numitrons are very dim. To reach full brightness, each Numitron segment must be driven in a sustained manner. [Side note: Yes, you *can* PWM a Numitron to adjust its brightness, see below.]

A better way to drive Numitrons is with shift-registers. Each Numitron is paired with its own 8-bit shift register, so that each bit corresponds to a display segment. A very commonly used shift register is the 74HC595. Since the Numitrons require 14mA per segment (112mA when all 8 segments are lit), a higher-current shift register is required. The TPICB595 is such a device and can sink up to 150mA *per output pin*.

Using a TPICBN595, the 3.3v power supply would be out-of-spec for our 2.5V Numitrons. We can either create a separate 2.5V supply or drop the voltage with a resistor. What resistance would be required to drop 3.3V to 2.5V at a current of 14mA? R = V/I = (3.3v-2.5V) / 14 mA = 56 ohms. The power required for each display segment would be the power dissipated by the resistor (0.8V * 14 mA = 11 mW) plus the display segment itself (35 mW) = 46mW. Each seven-segment display will require a shift-register and 7 current-limiting resistors. A six-digit display would therefore require 42 resistors.

I was about to use this approach when I discovered a very similar shift register with *built-in current limiting*, the TLC5916. This chip allows us to select the desired current (14mA) with a *single* resistor. Perfect! The output current is limited to the resistor value according to the chart at right. I chose a value of 1.5K, resulting in a segment current of 12.5mA.

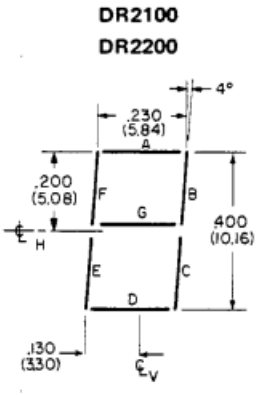
Rext	Output current (mA)
1.0 K	18.7 mA
1.2 K	15.6 mA
1.5 K	12.5 mA
2.2 K	8.5 mA
2.7 K	6.9 mA
3.3 K	5.7 mA
4.7 K	4.0 mA

Build a Numitron Display

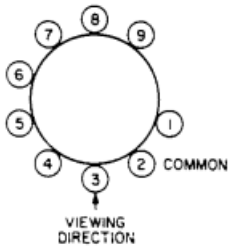
Time to build something! Let's wire a TLC5916 driver to a Numitron display, limit the current to 12.5mA/segment, and connect them to an MCU. For my project I am using an ESP8266 microcontroller module, the Wemos Mini D1, because it has built-in Wi-Fi that I intend to use later. But for now, any microcontroller will do.

The Numitron was not made for easy breadboarding. I created a small breakout board for that purpose.

Each segment in a seven-segment display has a name, ‘A’ through ‘G’, as shown in the diagram at right. The top segment is ‘A’. The segments continue in clockwise fashion through ‘F’. The middle segment is ‘G’.



The Numitron has 9 pins spaced 36-degrees apart. An empty slot at the 10th position helps identify which pin is #1. When looking at the front of the tube, pin 3 faces you and pin 8 is in the rear. *Bottom*-view diagram at right.



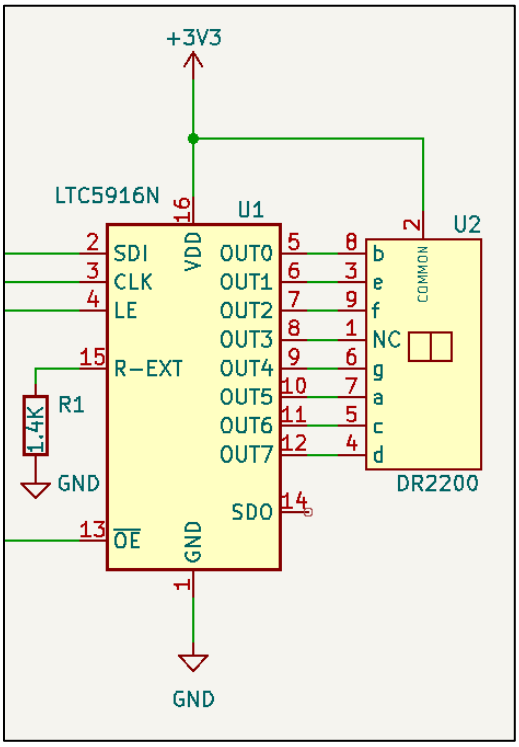
The table at right shows which pin corresponds to each segment. For example, Numitron pin 3 (the pin facing you) corresponds to segment ‘E’, which is bottom-left segment. Pin 6 is segment ‘G’, which is the middle segment. And so on.

Numitron Pin	Segment name	TLC5916 pin
1	(NC)	8
2	Common	(3.3V)
3	Seg ‘E’	6
4	Seg ‘D’	12
5	Seg ‘C’	11
6	Seg ‘G’	9
7	Seg ‘A’	10
8	Seg ‘B’	5
9	Seg ‘F’	7

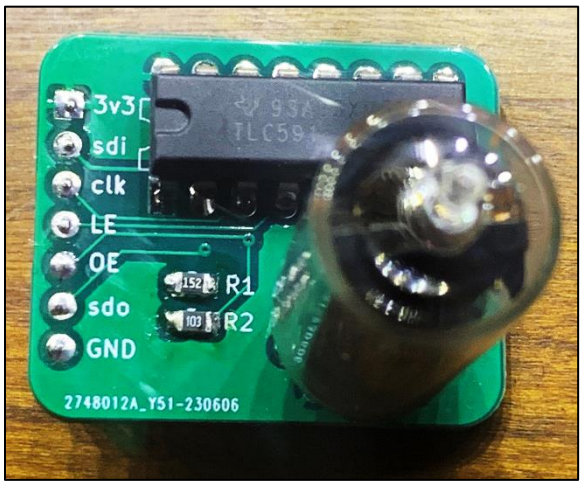
Connect the Numitron pins to a TLC5916 driver according to the third column in the table. Numitron pin 1 connects to TLC5916 pin 8, etc. Lastly, connect Numitron pin 2 ‘common’ to power, which is 3.3 volts in this case.

To complete the wiring, connect TLC5916 pin 16 to power and pins 1 and 13 to ground. Connect pin 15 to a 1.5K resistor and connect the other end of the resistor to ground.

Check your wiring against the schematic at right. The 3 lines labelled ‘SDI’, ‘CLK’, and ‘LE’ will be driven by a microcontroller. Pin OE goes to ground (not shown).



The [breakout board] contains all 3 schematic components. This board is handy for the experiments that follow.



Talking to a Numitron

The microcontroller and TLC5916 shift register communicate using the Serial Peripheral Interface (SPI). Data is sent from the microcontroller one bit at a time over the 'MOSI' data output line to the shift register data input line (SDI). It is synchronous data transfer, requiring the MCU's clock pulse (SCK) to be connected to the shift register clock line (CLK).

The shift register requires a third line called 'Latch Enable' (LE). An upgoing pulse on this line transfers the shift register contents to the output buffer. When LE returns low, the shift register outputs will be latched until the next LE pulse.

Here is a sketch that drives a Numitron, available on my GitHub site as Numitron_Step1:

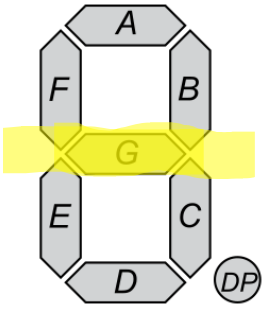
```
#include "SPI.h"              // Serial Peripheral Interface

#define LATCH     D8          // to TLC5916 LE (latch enable) pin

byte pattern = 0b00010000;    // bit4 = segment g  [map: dcagxfeb]


void writeByte(byte b) {
  SPI.transfer(b);            // send 1 byte to shift-register via SPI
  digitalWrite(LATCH,HIGH);   // latch it on upgoing pulse
  digitalWrite(LATCH,LOW);    // complete latch pulse
}

void setup() {
  pinMode(LATCH,OUTPUT);
  digitalWrite(LATCH,LOW);    // start with latch low
  SPI.begin();                // using SPI for data transfer
}

void loop() {
   writeByte(pattern);        // display segments according to pattern
   pattern = ~pattern;        // invert the pattern
   delay(1000);               // toggle at 1 second intervals
}
```

Starting from the top, SPI.h is the built-in Arduino library for the Serial Peripheral interface. On the ESP8266, the SPI bus uses pins D7 for data out and D5 for the clock. Our TLC5916 shift register requires a third data line to latch the data, for which we will use D8. So far, so good.

The writeByte() procedure is responsible for transferring data to the shift register. The highlighted line "SPI.transfer" sends the data, one bit at a time, from the MOSI pin (D7) to the shift register data input pin SDI. It does this by a protocol which both devices understand. When finished, bits 7 through 0 of the shift register mimick bits 7 through 0 of the data. But the output pins of the shift register will not reflect this new result until the latch pin has been pulsed high. This is accomplished by writing the Latch pin high, then low.

Looking back at the schematic, notice how 8 shift register data lines connect to 8 Numitron pins, one data line per segment. Illustrating this relationship in a slightly different way:

Seg 'D'	Seg 'C'	Seg 'A'	Seg 'G'	(dp)	Seg 'F'	Seg 'E'	Seg 'B'
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**



When a given bit is '1', the corresponding Numitron segment is lit; when the bit is 0, the segment is off. (Bit 3 connects to a decimal point if the Numitron tube contains one. Mine does not.)

How would one display a minus '-' sign? On a seven-segment display the central segment should be lit, which is segment 'G'. Set this bit to 1 and the remaining bits to 0:

Seg 'D'	Seg 'C'	Seg 'A'	Seg 'G'	(dp)	Seg 'F'	Seg 'E'	Seg 'B'
0	**0**	**0**	**1**	**0**	**0**	**0**	**0**

Therefore, the correct byte to send to the shift register is 0b0010000 (or 0x10 in hexadecimal). What character would we get if we lit every segment except for the center one? Yep, a zero:

Seg 'D'	Seg 'C'	Seg 'A'	Seg 'G'	(dp)	Seg 'F'	Seg 'E'	Seg 'B'
1	**1**	**1**	**0**	**1 (or 0)**	**1**	**1**	**1**

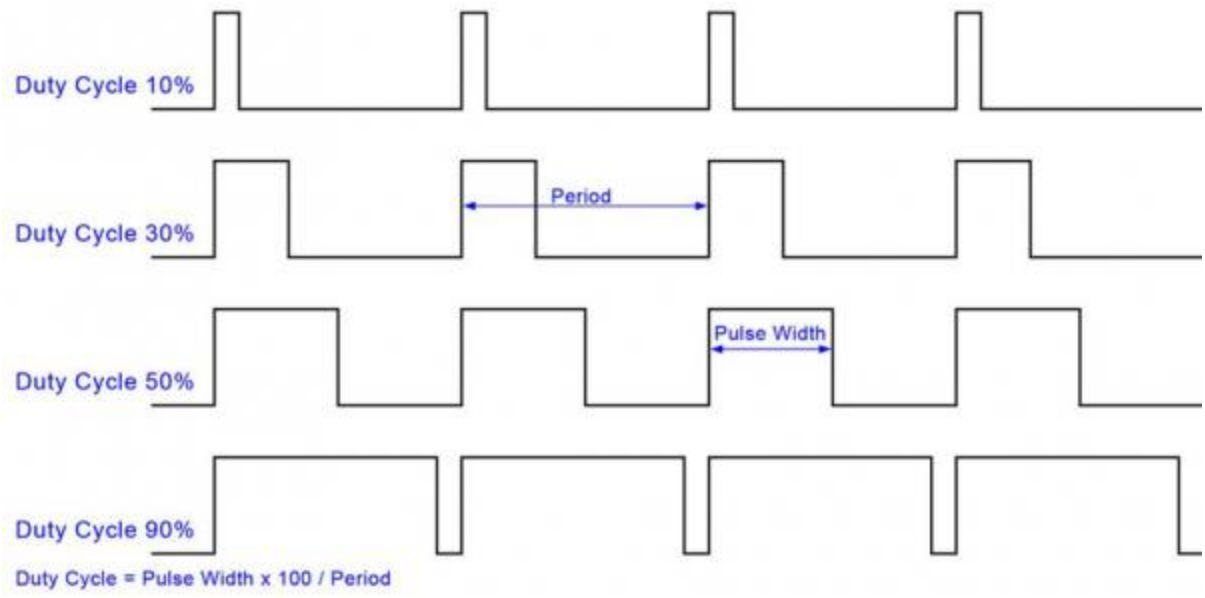
To create a zero, we should send 0b11101111 (0xEF hexadecimal) to the shift register. 0xE7 works equally as well, using a 0 in the decimal point position.

The loop() procedure toggles the Numitron display between '0' and '-' by flipping the data bits from '00010000' to their logical inverse: '11101111'. Feel free to experiment! For example, what pattern would '01100101' create? Answer: it makes a '7', and when the pattern is toggled to '10011010', it displays a lower-case 'c'.

Brightness Control

Numitron segments are like light-bulb filaments. They are bright at a certain voltage/current and get dimmer as the voltage/current is decreased (think flashlight with a dying battery). We can choose the brightness of our display by adjusting the current passing through each segment. The TLC5916 makes it easy: a 1.5K resistor fixes the current at a near-maximum 12.5mA. To make the display dimmer, reduce the current by choosing a larger resistor.

But how can we adjust brightness on the fly? We could replace the fixed resistors with potentiometers, one per tube. Fortunately, there is another way: pulse width modulation (PWM). With PWM, all the Numitron tubes are turned on/off at a frequency of several hundred kHz. We control the fraction of time in which the tubes are enabled. At a duty cycle of 10% (see below), each segment receives current for 10% of the time and appears to be off or very dim. At 90%, the segments are energized 90% of the time and are near full brightness.



The pin responsible for simultaneously controlling all segments is the TLC5916 OE (output enable) pin. Apply PWM to the OE pin and we can control brightness of the entire display.

The Arduino command for PWM is analogWrite(). It takes two parameters: the first is MCU pin to be controlled, and the second is the duty cycle (expressed as 0-255, where 255 is a duty cycle of 100%).

Consider the following example:

```
#define OE        D3          // to TLC5916 OE (output enable) pin

void setup() {
  pinMode(OE,OUTPUT);         // use OE for brightness control
}

void loop() {
  for (int brightness = 0; brightness < 255; brightness++) {
    analogWrite(OE, brightness);
    delay(10);
  }
  for (int brightness = 255; brightness >= 0; brightness--) {
    analogWrite(OE, brightness);
    delay(10);
  }
}
```

Download Numitron_Step2. The sketch defines D3 as the OE pin and configures it as an output. The program loop contains two for-loops. The first ramps up the brightness from 0 to 255, and the second ramps down the brightness from 255 to 0. Each loop calls on analogWrite() to set the duty cycle on pin D3 with 10mS between each change. The resulting PWM causes the display to alternatively brighten and dim every 256 x 10mS = 2.56 seconds.

Numitrons respond to PWM in a nonlinear fashion. My tubes do not light at duty cycles below 30%.

Let's do the Numbers

It's time to figure out how to display each digit. For each digit, consider which of the seven segments must be lit and construct a data byte representing that collection of segments. Digit "0" requires 6 illuminated segments and is represented by the byte 0xE7, as discussed above. We can do the same for the remaining digits, putting our results in a table.

To display the decimal point in devices that have them, add 8 (binary "1000") to the result.

Digit	d	c	a	g	x	f	e	b	= hex
0	1	1	1	0	0	1	1	1	0xE7
1	0	1	0	0	0	0	0	1	0x41
2	1	0	1	1	0	0	1	1	0xB3
3	1	1	1	1	0	0	0	1	0xF1
4	0	1	0	1	0	1	0	1	0x55
5	1	1	1	1	0	1	0	0	0xF4
6	1	1	1	1	0	1	1	0	0xF6
7	0	1	1	0	0	0	0	1	0x61
8	1	1	1	1	0	1	1	1	0xF7
9	1	1	1	1	0	1	0	1	0xF5
(blank)	0	0	0	0	0	0	0	0	0x00

Testing, testing, 1, 2, 3…

Create a sketch to count from 0 to 9, displaying each digit on the Numitron. The loop() procedure would look like this:

```
void loop() {
   for (int i=0; i<10; i++) {          // count 0..9
     displayDigit(i);                  // display each digit
     delay(1000);                      // at 1 second intervals
   }
}
```

displayDigit() should send segment data to the Numitron driver (the shift register) that corresponds to the desired digit, according to the table above. To do this, create an array called segments[] which contains the segment data for each digit:

```
const byte segments[] =
  {0xE7, 0x41, 0xB3, 0xF1, 0x55, 0xF4, 0xF6, 0x61, 0xF7, 0xF5};
```

Notice that segments[0] returns the segment data for digit 0; segments[4] returns data for digit 4, etc. With this array the displayDigit() routine is almost too easy:

```
void displayDigit(int i) {            // display a number 0..9 on the Numitron
  writeByte(segments[i]);             // send segment data to shift register
}
```

You should add error checking. For example, what happens when you call displayDigit(36)? But as an example, it works just fine. Here is the complete Numitron counter in less than 30 lines of code. Numitron_Step3:

```
#include "SPI.h"                      // Serial Peripheral Interface
#define LATCH     D8                  // to TLC5916 LE (latch enable) pin

const byte segments[] =
```

```
  {0xE7, 0x41, 0xB3, 0xF1, 0x55,       // segment data for digits 0..4
  0xF4, 0xF6, 0x61, 0xF7, 0xF5};       // segment data for digits 5..9

void writeByte(byte b) {
  SPI.transfer(b);                     // send 1 byte to shift register via SPI
  digitalWrite(LATCH,HIGH);            // latch it on upgoing pulse
  digitalWrite(LATCH,LOW);             // complete latch pulse
}

void displayDigit(int i) {             // display a number 0..9 on the Numitron
  writeByte(segments[i]);              // send segment data to shift register
}

void setup() {
  pinMode(LATCH,OUTPUT);
  digitalWrite(LATCH,LOW);              // start with latch low
  SPI.begin();                          // using SPI for data transfer
}

void loop() {
   for (int i=0; i<10; i++) {           // count 0..9
     displayDigit(i);                   // display each digit
     delay(1000);                       // at 1 second intervals
   }
}
```

The One-Digit Clock

Counting is nice, but we can do better: a one-digit clock. This fun and interesting timepiece requires no additional hardware.

One of my favorite time libraries is called "ezTime", by Rop Gonggrijp. Refer to my NTP Clock article for how I use this library. ezTime encapsulates all timekeeping functions, including obtaining time over the internet via NTP. To use it, include the library at the top of the sketch. The library assumes a working WiFi connection, so we will need to add the WiFi library as well:

```
#include <ezTime.h>                          // https://github.com/ropg/ezTime
#include <ESP8266WiFi.h>                     // use this WiFi lib for ESP8266
```

To access a local network, you must provide valid Wi-Fi credentials. Substitute your own Wi-Fi name (SSID) and password in the following defines:

```
#define WIFI_SSID  "yourWifiName"
#define WIFI_PWD   "yourWifiPassword"
```

With ezTime, synchronizing your microcontroller with UTC requires just two lines of code. First, in your setup routine, establish the Wi-Fi connection, then wait until the current time is returned via NTP. All the nitty-gritty details are handled internally by the ezTime library:

```
WiFi.begin(WIFI_SSID, WIFI_PWD);             // attempt WiFi connection
waitForSync();                               // wait for NTP packet return
```

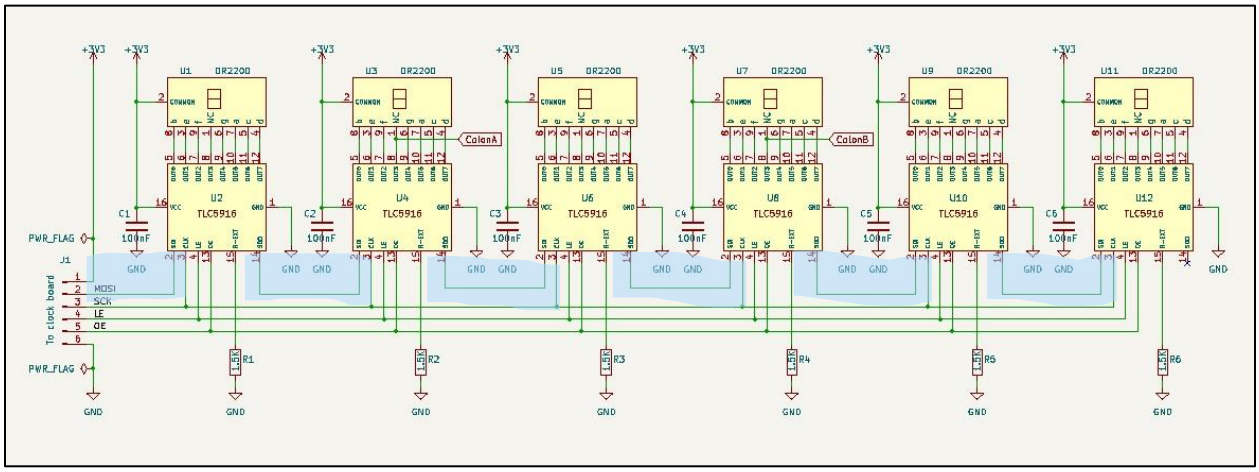
Displaying the time on a one-digit clock is as simple as displaying four sequential digits – two digits for the hour and two digits for the minute. Here is a simple solution:

```
void showTime(time_t t) {
  int h = hour(t);                          // get the current hour
  int m = minute(t);                        // and the current minute
  displayDigit(h/10);  delay(500);          // show hours as 2 digits
  displayDigit(h%10);  delay(500);
  displayDigit(m/10);  delay(500);          // show minutes as 2 digits
  displayDigit(m%10);  delay(500);
}
```

Now download and review the code for Numitron_Step4. It is a one-digit Numitron clock which automatically synchronizes to internet time – all in less than 100 lines of code.

Got lots of Numitrons? Try a 6-digit clock.

You might think that a 6-digit clock is more difficult than a 1-digit clock… but it's not. On the hardware side, you need five more Numitrons and five more TLC5916 shift registers. At first glance the schematic is imposing, but is no more than individual 6 tube-driver units.



Notice how the data-out and data-in lines are connected, highlighted in blue.

On the software side, we must write 6 bytes to the shift registers, one byte for each Numitron. For convenience, store the six bytes into an array called 'tubes'. tube[0] represents the first (left-most) Numitron and tube[5] is the last (right-most) one:

```
byte tube[6];                           // scratchpad memory for display tubes 0-5
```

Do an SPI transfer for each byte in the array. The right-most digit is sent first, because it must be "shifted" all the way through the shift-registers, from left to right, to the right-most tube. Therefore, the code below starts by sending data for tube[5] first and ending with tube[0]:

```
void writeDisplay() {
  for (int i=6; i>=0; --i)              // update digits 0-5 in reverse order
    SPI.transfer(tube[i]);              // send data to display, 1 byte/tube
  digitalWrite(LATCH,HIGH);             // latch it on upgoing pulse
  digitalWrite(LATCH,LOW);              // complete latch pulse
}
```

Finally, showTIme() determines time information for each tube position before calling writeDisplay(). Here is a sample implementation:

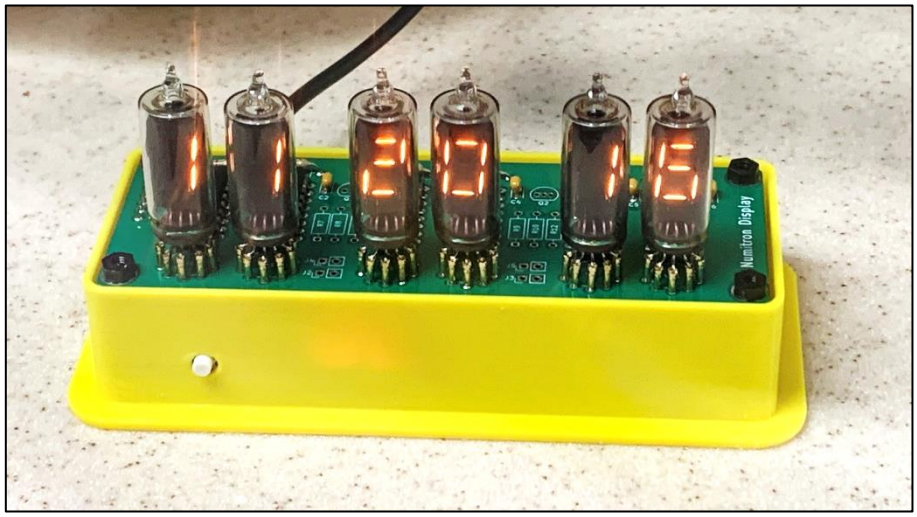
```
void showTime(time_t t) {                   // display time as "HH MM SS"
  int h = hour(t);                          // get hours, minutes, and seconds
  int m = minute(t);
  int s = second(t);
  tube[0] = segments[h/10];                 // hours 1st digit
  tube[1] = segments[h%10];                 // hours 2nd digit
  tube[2] = segments[m/10];                 // minutes 1st digit
  tube[3] = segments[m%10];                 // minutes 2nd digit
  tube[4] = segments[s/10];                 // seconds 1st digit
  tube[5] = segments[s%10];                 // seconds 2nd digit
  writeDisplay();                           // send data to display
}
```

Download Numitron_Step5, a simple but complete six-digit Numitron clock. Notice how little it differs from the 1-digit clock. In fact, it requires fewer lines of code!

But wait, there's more…

Of course, I couldn't stop there. I added the following features:

- WiFi configuration by mobile phone
- ASCII Time data output via USB
- Adjustable brightness level
- Selectable Local/UTC display
- Selectable date display
- Selectable 12/24-hr mode
- Leading zero suppression
- Alphanumeric support
- Diagnostic startup screen



I created two PCBs for my own clock: one for the power supply/MCU and one for the six-tube display. I also designed a 3D-printed base to house the boards. The PCB Gerbers and the final sketch, Numitron_clock, are on GitHub.

Last Updated: June 25, 2023