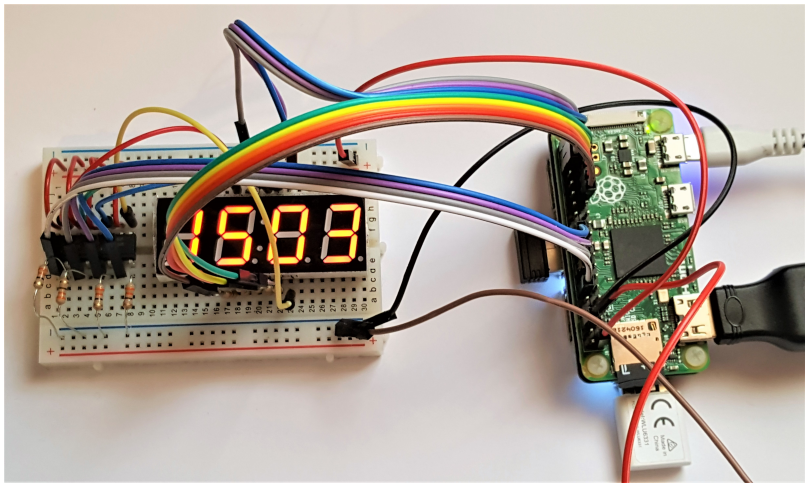
Raspberry Pi LED Clock and NTP Server

Jan 2017



Does this work?

Yes it does, so read on. At the time of writing I have finished the hardware (not got a PCB made yet, so am still using my prototype) and the Clock software is working. I have not yet started the NTP part, but will do so once the hardware is fully ready (I.E I have a PCB)

What is this for?

I like having a clock visible where I sit, I use an old tablet as a clock right now, which is fine. But I wanted something small enough to perch on the top of my monitor, or under it, that I could just glance at when I want to know the time.

I also want to know the clock is always correct, so I am going to connect it to the internet and sync it to one of the many super accurate time servers out their in the cloudy world of the internet. I also want to use this to set the time of my servers and client PCs at home. I have used Windows for this over the years, but it seems a bit flaky, and NTP just stops sometimes. Hence the super cheap and reliable Pi and Linux solution.

For about £10 I can build this little project, it will be fun and useful at the same time.

What you will need

A basic list of parts is:

1. Raspberry Pi (you can use any Pi for this, but if you use a Raspberry Pi 1, you will have to rewire)
2. SD Card, 4GB is enough if you use the minimal install (no GUI)
3. A wifi dongle or wired connection
4. A 7 Segment LED (This project uses common Anode)
5. A dozen 330R resistors
6. Two LTV-826 Opto Isolators (or equivalent)
7. Something to prototype on (breadboard)
8. Some wires (either use single core prototype wire or the jumper wires for Raspberry Pi's)
9. Some Python code to make the clock work (you can download what I am using)

I am using a Raspberry Pi Zero. The idea is that I can mount my 7 Segment LED (hopefully on a custom PCB eventually) on top of my Zero, to make a very small unit. While you can use a full Raspberry Pi 1, 2 or 3, this would be a total waste of money compared to the cost of the Pi Zero (unless you are going to use the Pi for other things like a web server for instance)

Hardware First

I am going to assume that you have a Raspberry Pi, and have the following:

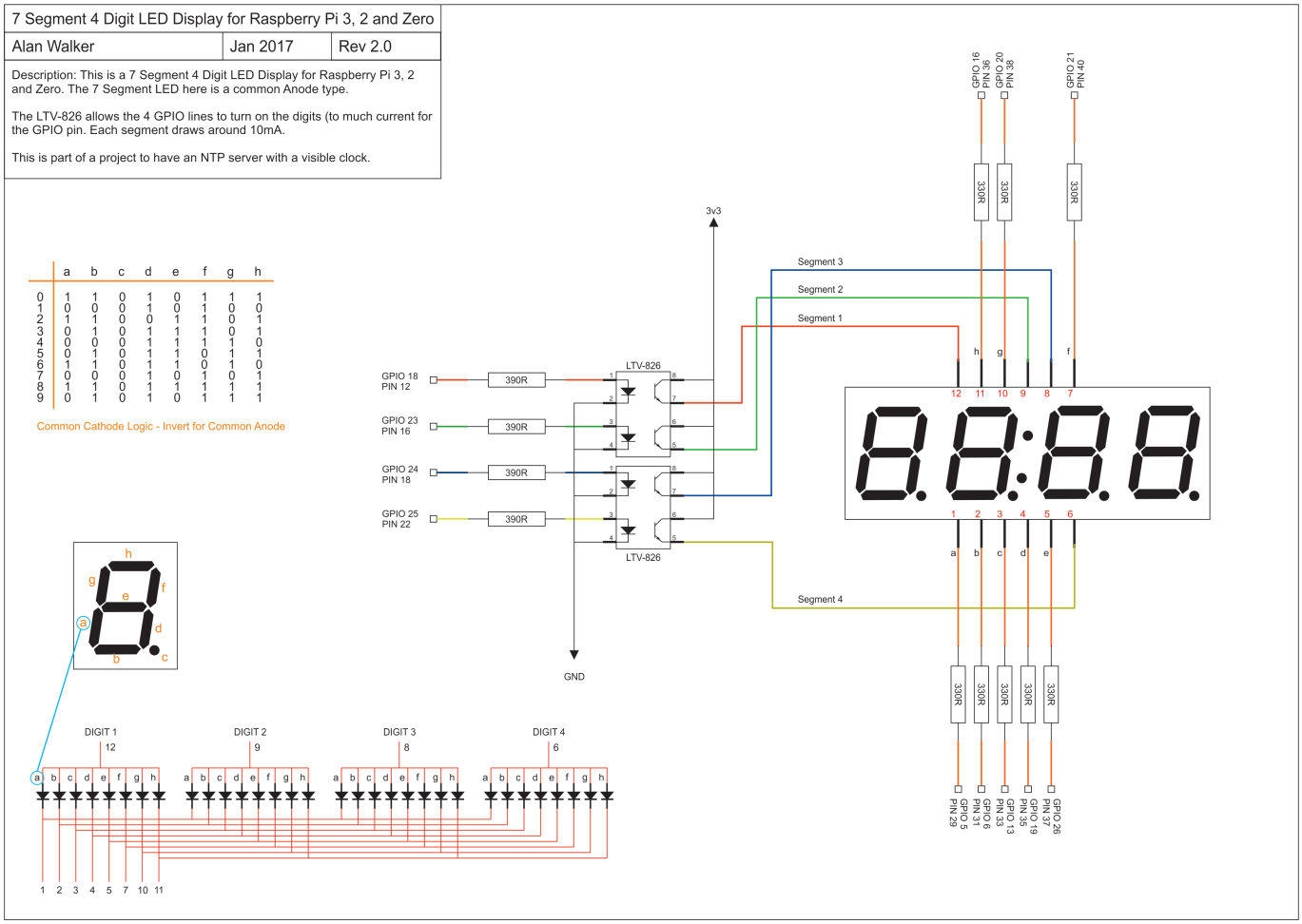
1. A working OS
2. An Installation of WiringPi
3. Working Python installation
4. Network Connectivity

Only when you meet the above requirements can you start this project (okay so the fourth one isn't actually required, you can just plug a keyboard and monitor in to the Pi)

I have decided to get the 7 Segment LED working first, then to tackle the NTP part second. This mainly because I just want to play with the 7 Segment LED, because it seems an interesting thing to do.

The Circuit

Let's take a look at the circuit for this project.



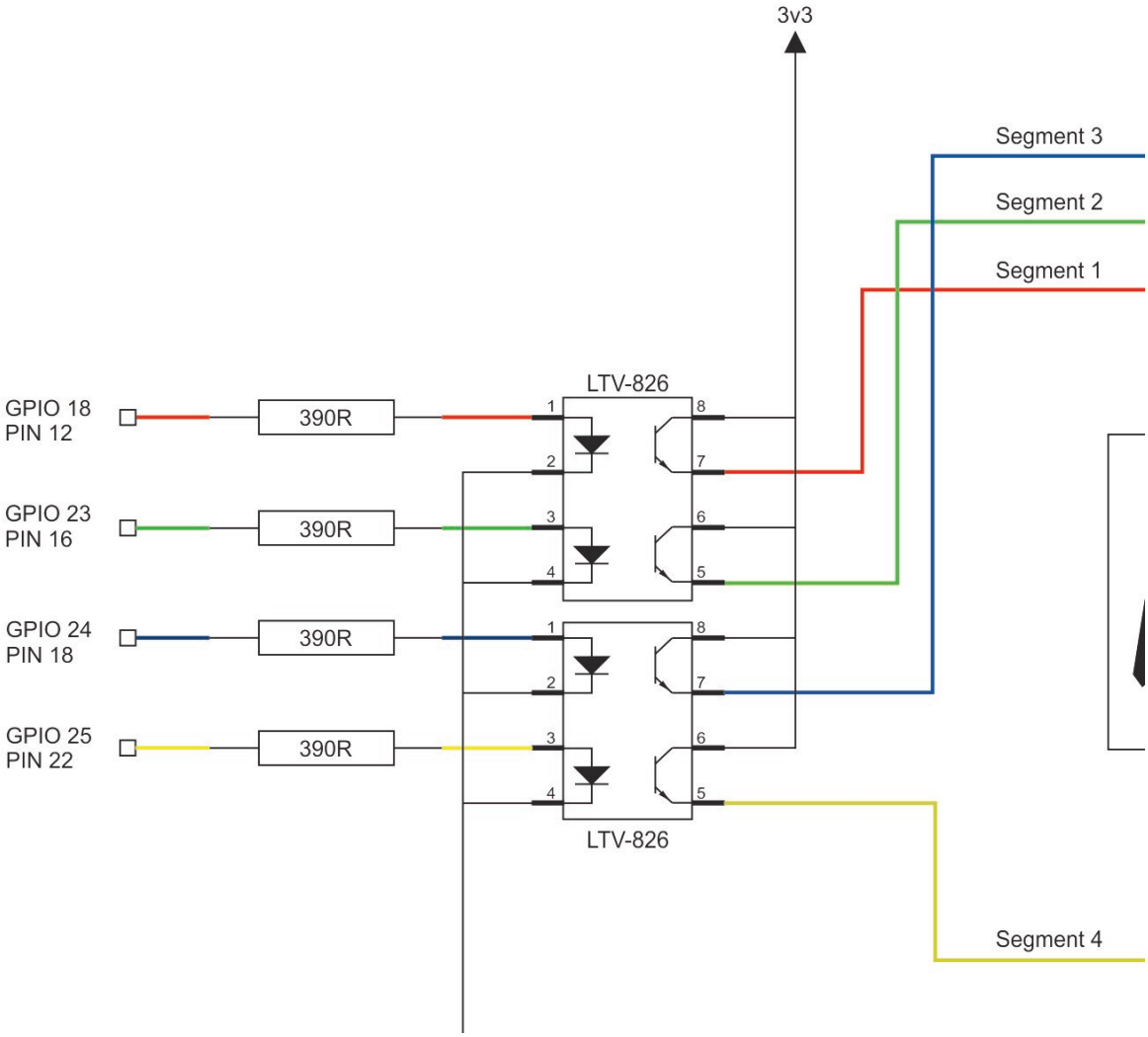
There are two Opto-Isolators to control the four digit lines. These are required because the Pi can't supply enough current (a whole 7 Segment digit using all segments requires around 40-50 mA, a GPIO pin should only sink or source around 16 mA).

There are 8 GPIO Pins connected to the 7 Segment LED via 330R resistors.

So in total there are only 12 connections to the Raspberry Pi.

Opto Isolators

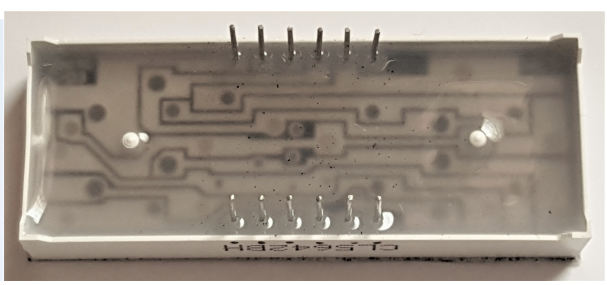
So what do these do exactly?



You can think of these as an electrical switch, operated by light. On the left hand side of the device, you supply a small current via a resistor (that is connected to a GPIO pin) and this turns on the LED in the package, this allows a current to flow on the right hand side, this current can be much greater than the input current. This stops too much current being pulled from the Raspberry Pi.

7 Segment LED

The 7 Segment LED I am using is a four digit version, with time indicator (the : in the middle) and there are many versions of these displays. Some have more pins than others, some are common Anode and some are Common Cathode. You can use Single Digit ones if you have them lying around, but you will have to common the pins together yourself.



Common Anode vs Common Cathode - All LEDs have a positive (Anode) and negative (Cathode) connection. On a common Anode display, all

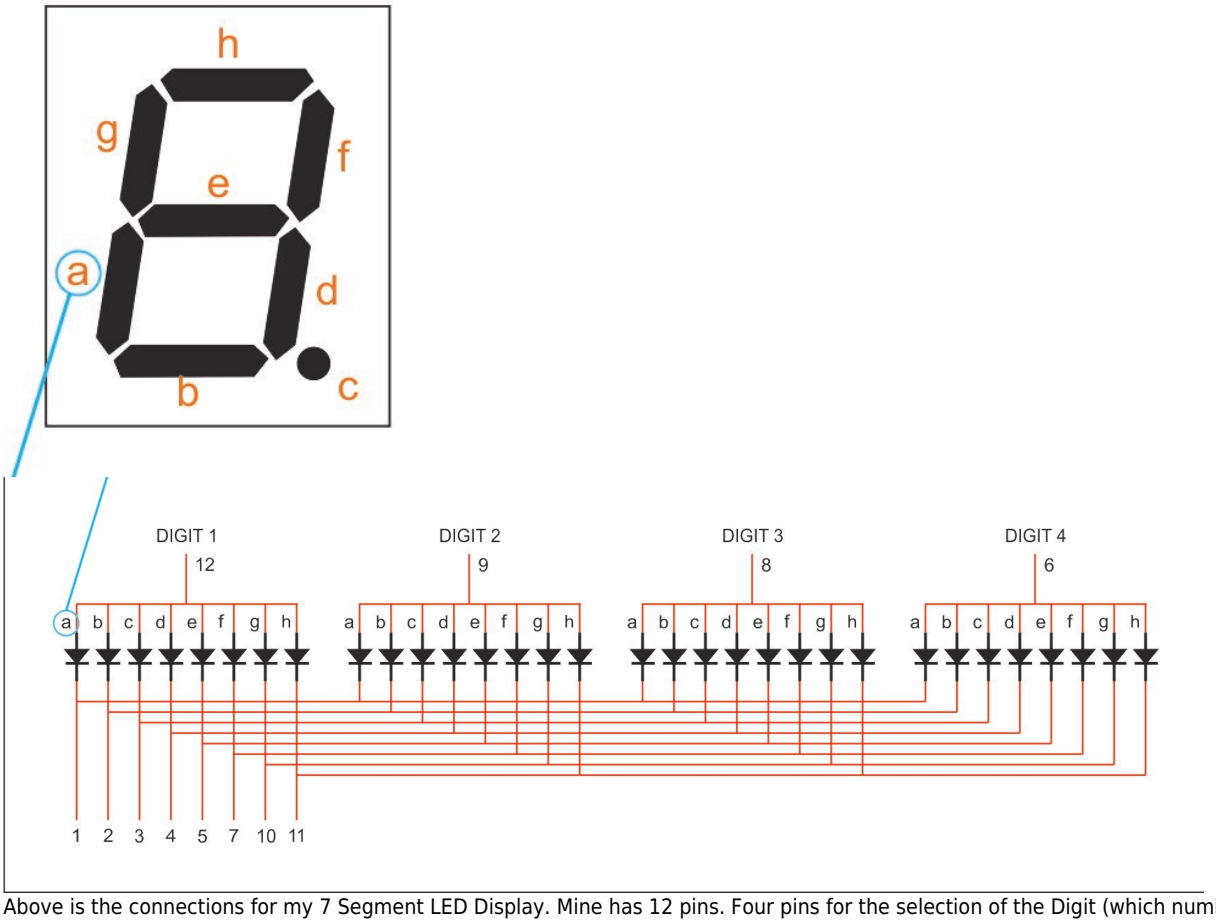
the Anodes (positives) are joined together, so you can only control the negative side of the LED segments. This means you switch each segment on by connecting or disconnecting the negative pins.

For Common Cathode, its is the opposite. All the negatives are ganged and connected to ground. All the positives are individually controlled.

So what? - Well if you use Common Cathode (which I have not) life is a little easier. You see with common Anode, you can only control the negative part of the LED, so to turn an LED on, you need to make the GPIO pin go to ground (or go low, or off whatever you prefer) This means an operation to turn an LED on (on in binary is generally accepted to be a 1) requires you make the pin go low (in binary that would be a 0).

So with common anode all of your logic is reversed. To make an '8' you would turn on all the segments, so 1111111, but because you have to make your pins go to ground on Common Anode, you actually have to set 0000000.

This project is using Common Anode if you want to use Common Cathode, that is fine, but you will have to change the wiring and the software.

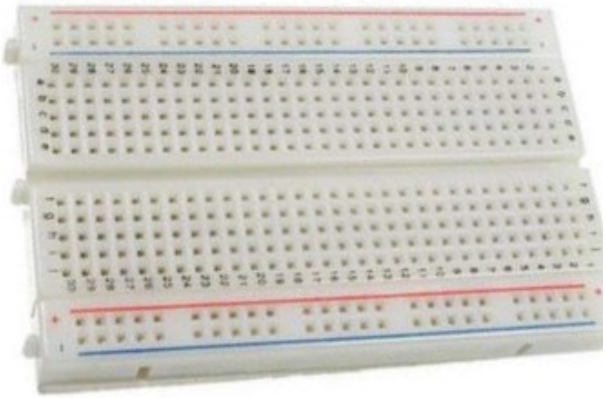


Above is the connections for my 7 Segment LED Display. Mine has 12 pins. Four pins for the selection of the Digit (which number is on) and 8 pins for the individual segments. There are 8 pins because this display has a full stop after each digit.

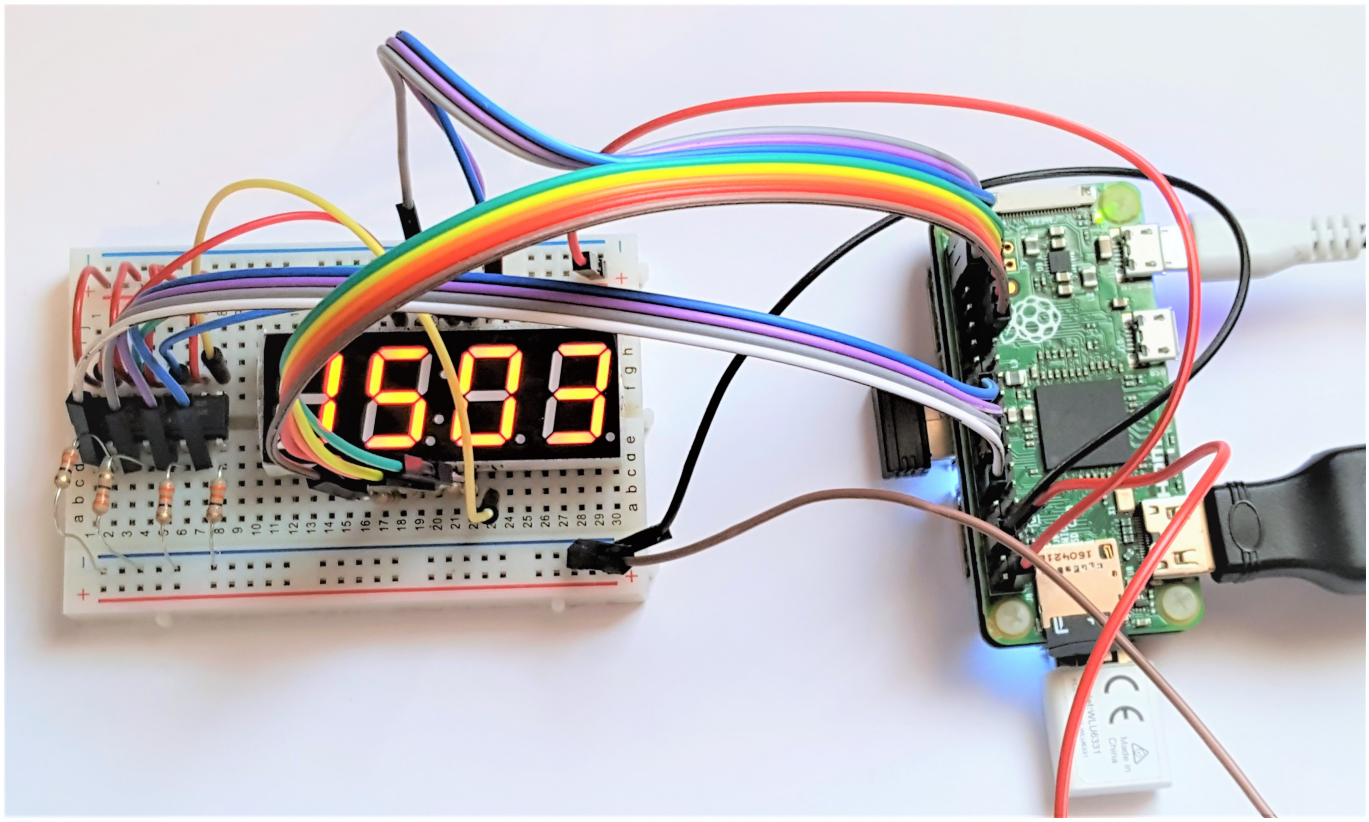
Connecting things up

For this project I have used some jumper wires I purchased from eBay a while ago when doing another project. There are three main types, Male to Male, Male to Female and Female to Female, as displayed here:

To test that everything will work as I think it will, I have used some breadboard to build my circuit on, these are really cheap from eBay (£2).



Below is my connected prototype build.



Once everything is connected we can do some basic testing.

How the Hardware Works

There are four digits on my display, and these can be enabled or disabled using the Pins 12(digit1), 9(digit2), 8(digit3) and 6(digit4).

The rest of the pins control which LED segment(s) are on.

Because the same pins control the segments on all the digits, we have to turn the digits on and off (or the same thing will display on all digits) So we enable digit one, set the segments we require to on (so a number is displayed) then we turn that digit off (and yes it goes off), then we repeat this process for digit2, digit3 and digit4.

To get the illusion that all four digits are always on, we do this process at a very high speed (100Hz) so that we can't see the segments going on and off.

Some Simple Testing

Before trying to get the Python Script working, it would be best to verify that everything is connected correctly and working as expected. We can do this from the command line using a utility called GPIO. GPIO comes part of some software called WiringPi.

First check to see if Wiring Pi is installed, use the following command.

```
gpio -v
```

If you get an error, you may have to install Wiring Pi:

```
sudo apt-get install wiringpi
```

Once installed try gpio -v again.

Now try using gpio readall:

```
gpio readall
```

```
 +-----+-----+---------+------+---+-Pi Zero--+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 |     |     |    3.3v |      |   |  1 || 2  |   |      | 5v      |     |     |
 |   2 |   8 |   SDA.1 |   IN | 1 |  3 || 4  |   |      | 5V      |     |     |
 |   3 |   9 |   SCL.1 |   IN | 1 |  5 || 6  |   |      | 0v      |     |     |
 |   4 |   7 | GPIO. 7 |  OUT | 0 |  7 || 8  | 1 | ALT0 | TxD     | 15  | 14  |
 |     |     |      0v |      |   |  9 || 10 | 1 | ALT0 | RxD     | 16  | 15  |
 |  17 |   0 | GPIO. 0 |   IN | 0 | 11 || 12 | 1 | OUT  | GPIO. 1 | 1   | 18  |
 |  27 |   2 | GPIO. 2 |   IN | 0 | 13 || 14 |   |      | 0v      |     |     |
 |  22 |   3 | GPIO. 3 |   IN | 0 | 15 || 16 | 0 | OUT  | GPIO. 4 | 4   | 23  |
 |     |     |    3.3v |      |   | 17 || 18 | 0 | OUT  | GPIO. 5 | 5   | 24  |
 |  10 |  12 |    MOSI |   IN | 0 | 19 || 20 |   |      | 0v      |     |     |
 |   9 |  13 |    MISO |   IN | 0 | 21 || 22 | 0 | OUT  | GPIO. 6 | 6   | 25  |
 |  11 |  14 |    SCLK |   IN | 0 | 23 || 24 | 1 | IN   | CE0     | 10  | 8   |
 |     |     |      0v |      |   | 25 || 26 | 1 | IN   | CE1     | 11  | 7   |
 |   0 |  30 |   SDA.0 |   IN | 1 | 27 || 28 | 1 | IN   | SCL.0   | 31  | 1   |
 |   5 |  21 | GPIO.21 |  OUT | 0 | 29 || 30 |   |      | 0v      |     |     |
 |   6 |  22 | GPIO.22 |  OUT | 0 | 31 || 32 | 0 | IN   | GPIO.26 | 26  | 12  |
 |  13 |  23 | GPIO.23 |  OUT | 1 | 33 || 34 |   |      | 0v      |     |     |
 |  19 |  24 | GPIO.24 |  OUT | 0 | 35 || 36 | 0 | OUT  | GPIO.27 | 27  | 16  |
 |  26 |  25 | GPIO.25 |  OUT | 0 | 37 || 38 | 0 | OUT  | GPIO.28 | 28  | 20  |
 |     |     |      0v |      |   | 39 || 40 | 1 | OUT  | GPIO.29 | 29  | 21  |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+-Pi Zero--+---+------+---------+-----+-----+
```

Now we can test the GPIO functionality. For this we will again use the GPIO command.

Warning The GPIO pins can be addressed using their GPIO number (so GPIO18 for example) or by their actual physical pin number (GPIO18 is on physical pin 12.

I always use what is reffered to as BCM numbers, which is the GPIO Number, not the physical pin number.

In the circuit diagram we can see that four GPIO lines control the Digits. These are:

```
GPIO18 - Digit1 - Connected to pin12
GPIO23 - Digit2 - Connected to pin16
GPIO24 - Digit3 - Connected to pin18
GPIO25 - Digit4 - Connected to pin22
```

We also have 8 GPIO lines to control each Segment. These are:

```
GPIO5  - Segment a - Connected to pin29
GPIO6  - Segment b - Connected to pin31
GPIO13 - Segment c - Connected to pin33
GPIO19 - Segment d - Connected to pin35
GPIO26 - Segment e - Connected to pin37
GPIO21 - Segment f - Connected to pin40
GPIO16 - Segment g - Connected to pin38
GPIO20 - Segment h - Connected to pin36
```

So first of all we will set the mode of the pins we are using to 'OUTPUT'

From the command line, enter the following: (the -g means we are using BCM GPIO numbers, if you leave off the -g you are using 'physical pin' numbers).

```
gpio -g mode 18 out
gpio -g mode 23 out
gpio -g mode 24 out
gpio -g mode 25 out
```

This sets the digit control pins to 'Output' mode. You probably won't see anything happen to your LEDs yet.

Now we can do the same for the GPIO Segment connections.

```
gpio -g mode  5 out
gpio -g mode  6 out
gpio -g mode 13 out
gpio -g mode 19 out
gpio -g mode 26 out
gpio -g mode 21 out
gpio -g mode 20 out
gpio -g mode 16 out
```

Next we will turn on all of the digits (by making the pins go 'high' binary '1') All we are doing setting the digits to active, nothing will happen until the next step when we start to turn on the LED Segments

```
gpio -g write 18 1
gpio -g write 23 1
gpio -g write 24 1
gpio -g write 25 1
```

Now we can start to to turn on the segments, I would do this one at a time by turning a segment on, then off and moving on to the next segment. This way you can tell if you are only turning on a single segment at a time.

Enter this line:

```
  gpio -g write 5 1
```

You should now have a single segment on all four digits.

If this has been successful, repeat the process for the remainder of the segments.

```
gpio -g write  6 1
gpio -g write 13 1
gpio -g write 19 1
gpio -g write 26 1
gpio -g write 21 1
gpio -g write 20 1
gpio -g write 16 1
```

Now all of your segments should be illuminated: