**20150329**

**NeoPixel BCD Clock!**

You may remember a while ago I did some demos with BCD encoders, and a made a BCD clock with a 8x8 LED matrix. We are revisiting some of this, but something I realized this time was I don’t think I ever explained what BCD is, and how we use it.

[20140620 BCD Demo using 10 bit Mini Level switch](https://docs.google.com/document/d/13fV_I7DV21LIn1pPRCs5ZzCYRnEkiSyuc5gOPlUj7Fw/edit?usp=sharing)

[20140622 Binary Coded Decimal Clock part 1](https://docs.google.com/document/d/1hOXjZ2GuDiXQTBj8BWKp-t5Jr5Lldvk3atKfLx4DR3g/edit?usp=sharing)

[20140624 BCD Clock Part 2](https://docs.google.com/document/d/13p2UyrSfoZ3rW-UaLGX936qSvs2C7vP2TLTBV09A-_Y/edit?usp=sharing)

To understand this we kind of need to look at number systems in general,

BASE 10 - Decimal

We call it base 10 because there are 10 numbers before you get to the number 10. In other words Zero to Nine (Single Digits) This is the system we are most use to it is what we learn in school.

Each digit represents a place value.

| Thousand | Hundred | Tens | Ones |
| --- | --- | --- | --- |
| 3 | 4 | 2 | 6 |

So this is 3426 - or we have 3 thousands, 4 hundreds, 2 tens, and 6 ones

The same is true for other base units - For example Base 2 - has two numbers, zero and one.

We call this system Binary.

In Binary, we have place values(most of the time called Bits) as well, they are:

| 2048 | 1024 | 512 | 256 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |

These numbers above the 4 place value should look familiar, and this is no mistake, computers work with binary, and you’ll normally see memory and hard-drive space in some multiple of the above.

110101100010 base 2 is 3426 base 10 How did we get there?

We have 1 - 2, 1 - 32, 1 - 64, 1 - 256, 1 - 1024, and 1 - 2048 if we add those numbers together we should get 3426.

2 + 32 + 64 + 256 + 1024 + 2048 = 3426

Currently we have 12 bits, and a lot of A2D converters use 12 bit resolution - what this is mean is the largest analog number they can convert is 4095, you’ll also see 16 bit, 32 bit and 64 bit.

| 32768 | 16384 | 8192 | 4096 | 2048 | 1024 | 512 | 256 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

This above is the bit values for 16 bit binary - in 16 bit A2D you can have a number of 65535 - this number should also look familiar to you - it’s the largest integer value an Arduino can have.

So more bits means bigger numbers. (Just for instance here, the next number in series would be 65536, and just that one bit will increase the largest number to 131071) Interesting huh?

So this video explains a pretty simple way to convert from Base 10 to Base 2 - He doesn’t explain the place values and all that, but if you watch, he does use the place values in his checks.

<https://youtu.be/Q2UgMYwWiO4> There are other ways to convert the number, most good scientific calculators will do some base convertions.

In the computer world, and micro controller world you’ll also see Base 16 numbers (also called Hex or Hexadecimal)

Hex is 16 numbers, and uses letters as numbers. (Zero to F)

| Hex | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Dec | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |

Hex can be a little confusing at first. But as you can see in the chart above a Dec 15 is a Hex F

Normally and most of the time in the computer world you see Hex expressed with 2 digits -

0F sometimes it is preceded by a number sign, sometimes by a 0x

Many times with dealing with RGB values, you’ll see Hex in this format: 0xFFFFFF - This is still 2 digits (really it is) FF = 255 or in the Arduino world max bright for a color R, G, B

Just like other bases - Hex as place values - let’s look at our number 3426 base 10 in Base 16

| 1048576 | 65536 | 4096 | 256 | 16 | 1 |
| --- | --- | --- | --- | --- | --- |
|  |  |  | D | 6 | 2 |

So - As you can see with just a few digits of Hex you can express very large numbers. Let’s verify that we can convert the Hex to the number we expect.

We have 256 \* D (13) = 3328, We have 6 \* 16 = 96, and we have 2 times 1 = 2

3328+96+2 = 3426 base 10.

So here is another video that explains how to convert Base 10 to Base 16 a bit.

<https://youtu.be/iVDZ23vzIWQ>

So that is a very basic idea of using different bases, There are other bases in use as well, but these are the two you’ll see the most in computers/micro-controlers.

But I haven’t explained anything at all about BCD - Binary Coded Decimal - BCD is a special case of Binary, It is used for encoding.

So what is encoding?

Encoding allows the perceived item of use or interest to be converted into a construct that can be stored with the brain, and recalled later from short term or long term memory.

Doesn’t help really does it. Ok, for us, Encoding is taking a Decimal number and storing each digit as a Binary number represented by a fixed number of bits (Normally this will be 4 or 8 bits) and Special Bit patterns are sometimes used for a sign or for other indications (error or overflow).

So lets look at our example again. 3426 base 10

We know in Base 2 it’s 110101100010 base 2

We also know what 4 Bit binary would look like.

| Dec Value | 8 | 4 | 2 | 1 |
| --- | --- | --- | --- | --- |

In BCD we take each digit, and convert that to binary using the above.

3 4 2 6

Starting at the most significant digit 3 would convert to 0011

Next 4 would convert to 0100

Next 2 would convert to 0010

Next 6 would convert to 0110

So 3426 base 10 would be 0011 0100 0010 0110 BCD

You can write this number altogether if you wish, we know that each 4 bits is a number, so

0011010000100110 BCD = 110101100010 base 2 = 3426 base 10 = D62 base 16

When using BCD it is very important to know how many bits you are using (4 or 8 bit) for the encoding, it is also very very important to know if you are starting with the Most Significant Bit or the Least Significant Bit, as each group of bits represents one digit this could change the number.

You also have to have all the bits, even if they are zero, so 0000 in 4 bit BCD is zero, 0 in 4 bit BCD is invalid. Same is true for 8 bit BCD. All bits must be present. The benefits to using BCD is speed of converting from Base 10 or 16 to BCD, The cons are a small increase in the complexity of circuits needed to implement basic arithmetics and slightly less dense storage (if you remember back to the Demo of BCD switches I have, you’ll remember how many data lines I had to use to get 3 digits into the Arduino - I think it was all 16)

BCD really can only go from zero to 9, as each digit of a decimal number is separated from the other digits.

So.

| Decimal Digit | BCD Digit |
| --- | --- |
| 0 | 0000 |
| 1 | 0001 |
| 2 | 0010 |
| 3 | 0011 |
| 4 | 0100 |
| 5 | 0101 |
| 6 | 0110 |
| 7 | 0111 |
| 8 | 1000 |
| 9 | 1001 |

So that is the very basics of BCD, now that I have that out of the way, Lets look at the clock,

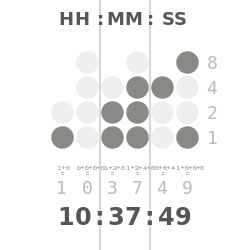
So, let’s think about how we can use this for a clock. Think about how time works, we are on a

we have 24 hours that are broken down to 60 minutes, each minute is broken down into 60 seconds.

As far as the seconds and minutes go, we never “see” 60, the cycle goes from 00 to 59. So for the 1st digit of each (minutes and seconds) we can get away with just displaying 3 bits.

We need to decide if we want to display 24 hour time or if we just want the time to display on 12 hours. Either way for the 1st digit of the hour we can get away with displaying 1 or 2 bits.

So we will come up with something like this.

The clock I made I left the missing (from here) 2 bits as LEDs, they are not being use, but I had thought about doing more than just a clock with this project. I also decided that I didn’t need to know 24 hour time.

The previous BCD clock I did was using a DS1370 RTC, and a 8x8 matrix in that project, the DS1370 sends binary to the arduino when it is asked for a time, that binary needed to be converted to BCD. And I was able to just send the BCD to the correct row of the 8x8 matrix, I sent the whole 4 bit BCD to it, so it would turn on and off leds as needed.

In this project, I am displaying BCD, I am using Neopixel LEDs (WS2811/WS2812), and the Arduino YUN, there is no RTC in this project.

<https://codebender.cc/sketch:98788> - Codebender doesn’t support the YUN at this time, but if you copy the code to your compiler you should beable to use it.

When the YUN gets power and connects, the linux side of things get the date from the time server, and keeps fairly good time while it’s running, I couldn’t tell if it was running a little fast, or if the time was a little fast from the time server, or if my computer just has the wrong time. It’s pretty close.

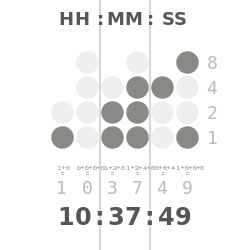
On the AVR side of things, one of the things it can do is get the time using the bridge. The time is returned as a decimal number. And is returned in 24 hour time.

So I do some math if the hour is above 13 it will subtract 12 and return a hour that should be from 1 to 12.

I also print the time to the serial monitor, just as a check.

More math happens, and the digits are separated. So We end up with the Least Significant number for hours minutes and seconds. (What do this mean?)

Lets look at this again:

10:37:49

secondUnit = ((seconds/10\*16)+(seconds%10))%10;

minuteUnit = ((minutes/10\*16)+(minutes%10))%10; //sets the variable minuteUnit and hourUnit for the unit digits

hourUnit = ((hours/10\*16)+(hours%10))%10;

using the above we would get:

secondUnit = 9

minuteUnit = 7

hourUnit = 0

For this we don’t care about the most significant digit, we take care of that in another way.

I am using 21 NeoPixels, neopixels are number from 0 to 20, The 0 neopixel would be the LSD of the seconds (it’s the 1 place bit of the LSD of seconds) Neopixel 3 would be the 8th value of the LSD of seconds.

For the next steps we have to look at our BCD chart above, what we are going to do is use if or statements to figure out which LEDs should be turned on for the LSD of each of our groups.

We also need to know which neopixel to turn on. So let’s look:

Here is what we know.

Dec 1 = 0001 BCD

Dec 2 = 0010 BCD

Dec 3 = 0011 BCD

Dec 4 = 0100 BCD

Dec 5 = 0101 BCD

Dec 6 = 0110 BCD

Dec 7 = 0111 BCD

Dec 8 = 1000 BCD

Dec 9 = 1001 BCD

Dec 1, Dec 3, Dec 5, Dec 7, Dec 9 all have bits in common - the ones place bit.

Dec 2, Dec 3, Dec 6, Dec 7 all have bits in common - the twos place bit.

Dec 4, Dec 5, Dec 6, Dec 7 all have bits in common - the fours place bit.

and only Dec 8 and Dec 9 share the 8th place bit.

So Neopixel Zero (0) is the ones place, we will check to see if the LSD is any of the following

Dec 1, 3, 5, 7 or 9 - and turn Neopixel Zero on, if the LSD isn’t any of these digits we will turn the zero neopixel off.

Neopixel 1 is the twos place, and we check to see if the LSD is any of the following, DEC 2, 3, 6 and 7 and we will turn on Neopixel 1, if it’s not any of these digits we will turn off Neopixel 1

neopixel 2 is the fours place, and we check to see if the LSD is any of the following DEC 4, 5, 6 and 7 we will turn on Neopixel 2, if it’s not any of these numbers we will turn off Neopixel 2.

Neopixel 3 is the 8th place, and we check to see if the LSD is either Dec 8 or 9, turning on Neopixel 3, if it’s not either one we will turn off Neopixel 3.

With those 4 simple IF/OR statements, we have just turned on the LSD of the seconds.

We do something a little different for the MSD of the seconds.

MSD digital of seconds uses NEOPixel 4, 5, 6 and 7 (thou 7 isn’t used, but is wired up)

For this we look at the MSD by using a combination of IF AND and IF OR statements

We are dealing with decimal number between zero and five.

In this case we don’t care about the LSD, so we just look to see if MSD is larger or equal to 10 but smaller than 20, OR if MSD is larger or equal to 30 but smaller than 40, or MSD is greater than or equal to 50 and less than 60 we will turn on Neopixel 4, if none are true we will turn off Neopixel 4.

We also need to check for other IF MSD is greater or equal to 20 and less then 40 We turn on NeoPixel 5 if not we turn off NeoPixel 5.

Finally we check to see if it’s greater than or equal to 40 but less than 60 We turn on Neopixel 6 if not we turn off Neopixel 6.

For the Minutes, we repeat the above, but use different Neopixels to turn on or off.

Neopixel 8 is the ones place for the LSD minutes, 9 would be the twos place, 10 would be fours place, etc.

Minutes MSD work the same way, just changing out the neopixel numbers.

For the LSD of the hours, we can use the same type of IF OR statements, changing which neopixel to turn on or off.

For the MSD of the hours, we only need to find out if it is greater or equal to 10 and less than 20, and if we are using 24 hour time, if it is greater than or equal to 20 and less than 24, turning on or off the correct Neopixels as needed.

Here is a video of it working:

<https://youtu.be/h8GHHtCCFLw>