

Charlieplexing LEDs- The theory

by [rgbphil](#) on May 21, 2007

Table of Contents

intro: Charlieplexing LEDs- The theory	2
File Downloads	2
step 1: Some LED theory	2
step 2: The Laws (of electronics)	3
step 3: Introducing 'complementary drive'	4
step 4: Finally....a Charlieplex matrix	5
step 5: Tri-states (not tricycles)	6
step 6: Some Practical matters	6
step 7: References	7
Related Instructables	7
Advertisements	7
Customized Instructable T-shirts	7
Comments	7

intro: Charlieplexing LEDs- The theory

This instructable is less a build you're own project and more a description of the theory of charlieplexing. It's suitable for people with the basics of electronics, but not complete beginners. I've written it in response to the many questions I've gotten in my previously published Instructables.

What is 'Charlieplexing'? It is driving lots of LEDs with only a few pins. In case you're wondering Charlieplexing is named after Charles Allen at Maxim who developed the technique.

This can be useful for lots of things. You may need to display status information on a small microcontroller, but only have a few pins spare. You may want to show a fancy dot matrix or clock display but don't want to use lots of components.

Some other projects demonstrating charlieplexing you may want to look at are:

How to drive a lot of LEDs from a few microcontroller pins.

by Westfw :- <http://www.instructables.com/id/ED0NCY0UVWEP287ISO/>

And a couple of my own projects,

The Microdot watch:- <http://www.instructables.com/id/EWM2OIT78OERWHR38Z/>

The Minidot 2 clock:- <http://www.instructables.com/id/E11GKKELKAEZ7BFZAK/>

Another cool example of the use of charlieplexing is at:

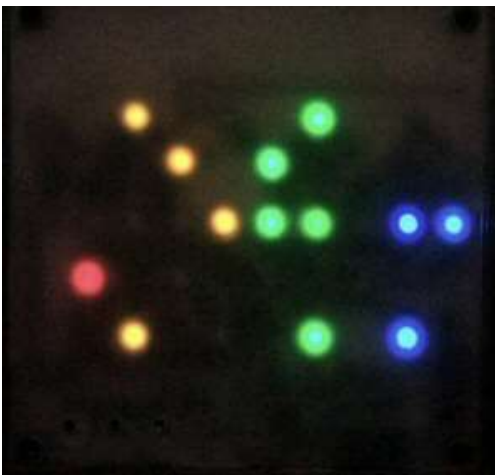
<http://www.jsdesign.co.uk/charlie/>

The Minidot 2 clock introduces an advanced charlieplexing scheme for fading/dimming which won't be discussed here.

UPDATE 19 August 2008 : I've added a zip file with a circuit that may be able to exploit the matrix charlieplexing for high power LEDs discussed (at length :)) in the comments section. It has a pushbutton + position encoder to do a user interface, plus circuitry for either USB or RS232 computer control. Each of the high side voltage rails can be set to one of two voltages, say 2.2V for RED LEDs and 3.4V for green/blue/white. The voltage for the high side rails can be set by trimpot. I'd envisage that a 20wire IDC ribbon cable be plugged into the board, and 20pin IDC connectors added along the length of the ribbon, each LED board having links to whatever wires in the matrix are desired. The circuit is in Eagle Cad and rendered in the sub image below. The high side circuit is implemented by using optocouplers which I think might be suitable.

I haven't actually tested this circuit nor written any software because of lack of time, but have put it up for comment, I'm particularly interested in the optocoupler implementation. Anyone brave enough to give it a go...please post your results.

UPDATE 27th August 2008: For those not using EagleCad....added below is a pdf of the schematic



File Downloads



HPCharlie.zip (968 KB)

[NOTE: When saving, if you see .tmp as the file ext, rename it to 'HPCharlie.zip']



HPCharlie.pdf ((595x842) 44 KB)

[NOTE: When saving, if you see .tmp as the file ext, rename it to 'HPCharlie.pdf']

step 1: Some LED theory

Charlieplexing relies on a number of useful aspects of LEDs and modern microcontrollers.

Firstly what happens when you connect an LED to electricity.

The main diagram below shows what is called the I_f v V_f curve of a typical 5mm low power LED.

I_f stands for 'forward current'

V_f stands for 'forward voltage'

The vertical axis in otherwords shows the current that will flow through an LED if you put the horizontal axis voltage across it's terminals. It works the other way around as well, if you measure that the current is of some value, you can look across to the horizontal axis and see the voltage the LED will present across it's terminals.

The second diagram shows a schematic representation of an LED with I_f and V_f labelled.

From the main diagram I've also labelled areas of the graph that are of interest.

- The first area is where the LED is 'off'. More accurately the LED is emitting light so dimly you won't be able to see it unless you had some sort of super-duper image

<http://www.instructables.com/id/Charlieplexing-LEDs--The-theory/>

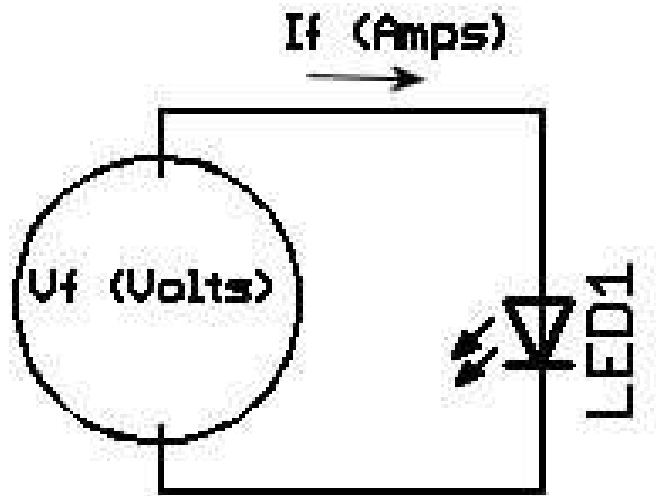
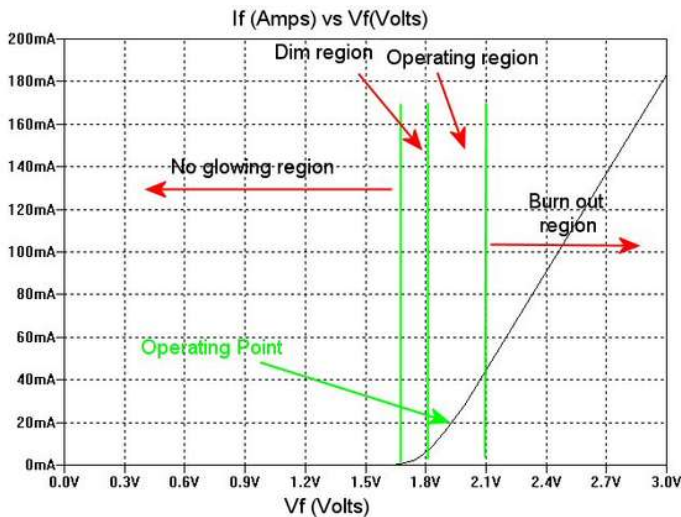
- intensifier.
- The second area has the LED just slightly emitting a dim glow.
 - The third area is where an LED is usually operated and is emitting light at the manufacturers rating.
 - The fourth area is where an LED is operated beyond its operating limits, is probably glowing very brightly but alas for only a short time before the magic smoke inside escapes and it won't operate again.....ie in this area it burns out because too much current flows through it.

Note that the I_f/V_f curve or operating curve of the LED is a 'non-linear' curve. That is, it is not a straight line...it has a bend or kink in it.

Lastly this diagram is for a typical 5mm red LED designed to operate at 20mA. Different LEDs from different manufacturers have different operating curves. For example in this diagram at 20mA the forward voltage of the LED will be approximately 1.9V. For a blue 5mm LED at 20mA the forward voltage might be 3.4V. For a high power white Luxeon LED at 350mA the forward voltage might be around 3.2V. Some LEDs packages might be several LEDs in series or in parallel, changing the V_f/I_f curve again.

Typically a manufacturer will specify an operating current which is safe to use the LED at, and the forward voltage at that current. Usually (but not always) you get a graph similar to below in the datasheet. You need to look at the datasheet for the LED to determine what the forward voltage is at different operating currents.

Why is this graph so important? Because it shows that when a voltage is across the LED, the current that will flow will be according to the graph. Lower the voltage and less current will flow.....and the LED will be 'off'. This is part of the theory of charlieplexing, which we'll get to in the next step.



step 2: The Laws (of electronics)

Still not yet at the magic of charlieplexing yet....we need to go to some basics of electronics laws.

The first law of interest states that the total voltage across any series of connected components in an electrical circuit is equal to the sum of the individual voltages across the components. This is shown in the main diagram below.

This is useful when using LEDs because your average battery or microcontroller output pin will never be exactly the right voltage to run your LED at the recommended current. For example a microcontroller will typically run at 5V and its output pins will be at 5V when on. If you just connect an LED to the output pin of the micro, you'll see from the operating curve in the previous page too much current will flow in the LED and it will get hot and burn out (probably damaging the micro as well).

However if we introduce a second component in series with the LED we can subtract some of the 5V so that the voltage left is just right to run the LED at the proper operating current.

This is typically a resistor, and when used in this way is called a current limiting resistor. This method is used very commonly and leads to what is called 'ohms law'....so named after Mr Ohm.

Ohms law follows the equation $V = I * R$ where V is the voltage that will appear across a resistance R when a current I is flowing through the resistor. V is in volts, I is in amps and R is in ohms.

So if we have 5V to spend, and we want 1.9V across the LED to get it to run at 20mA then we want the resistor to have $5 - 1.9 = 3.1V$ across it. We can see this in the second diagram.

Because the resistor is in series with the LED, the same current will flow through the resistor as the LED, ie 20mA. So rearranging the equation we can find the resistance we need to make this work.

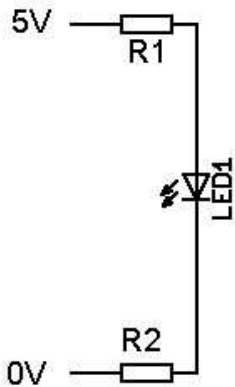
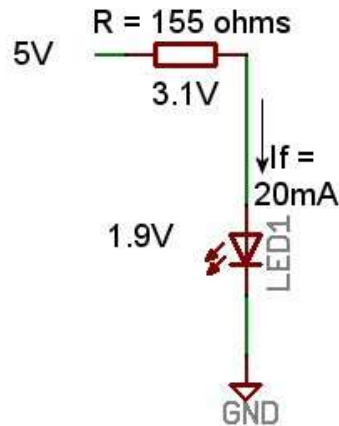
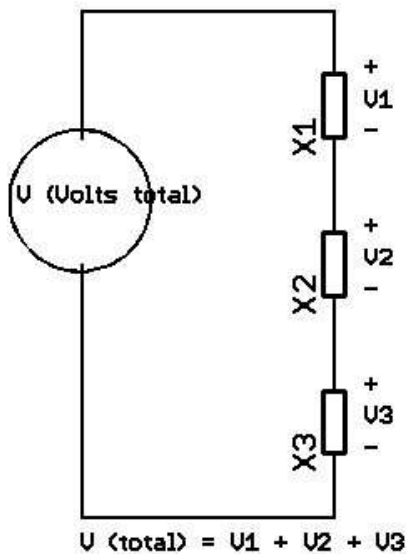
$V = I * R$
 so
 $R = V / I$
 substituting the values in our example we get:
 $R = 3.1 / 0.02 = 155\text{ohms}$
 (note 20mA = 0.02Amps)

Still with me so far...cool. Now look at diagram 3. It has the LED sandwiched between two resistors. According to the first law mentioned above, we have the same situation at the second diagram. We have 1.9V across the LED so it is running according to its spec sheet. We also have each resistor subtracting 1.55V each (for a total of 3.1). Adding the voltages together we have $5V$ (the microcontroller pin) = $1.55V$ (R1) + $1.9V$ (the LED) + $1.55V$ (R2) and everything balances out. Using ohms law we find the resistors need to be 77.5 ohms each, which is half the amount calculated from the second diagram.

Of course in practice you'd be hard pressed to find a 77.5ohm resistor, so you'd just substitute the nearest available value, say 75ohms and end up with a little more current in the LED or 82ohms to be safe and have a little less.

<http://www.instructables.com/id/Charlieplexing-LEDs--The-theory/>

Why on earth should we be doing this resistor sandwich to drive a simple LED.....well if you have one LED it's all a bit silly, but this is an instructable on charlieplexing and it comes in handy for the next step.



step 3: Introducing 'complementary drive'

Another name that is more accurate to describe 'charlieplexing' is 'complementary drive'.

In your average microcontroller you can in firmware tell the micro to set an output pin to be either a '0' or a '1', or to present a 0V voltage at the output or a 5V voltage at the output.

The diagram below now shows the sandwiched LED with a reversed partner....or a complement LED, hence complementary drive.

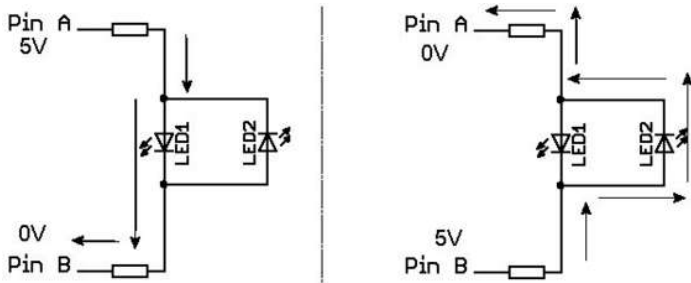
In the first half of the diagram, the micro is outputting 5V to pin A, and 0V to pin B. The current will thus flow from A to B. Because LED2 is oriented backwards to LED1 no current will flow through it and it will not glow. It's what is called reverse biased. We have the equivalent of the situation in the previous page. We can basically ignore LED2. Arrows show the current flow.

An LED is essentially a diode (hence Light Emitting Diode). A diode is a device that allows current to flow in one direction, but not in the other. The schematic of an LED sort of shows this, current will flow in the direction of the arrow.....but is blocked the other way.

If we instruct the micro to now output 5V to pin B and 0V on pin A we have the opposite. Now LED1 is reverse biased, LED2 is forward biased and will allow current flow. LED2 will glow and LED1 will be dark.

Now might be a good idea to look at the schematics of the various projects mentioned in the introduction. You should see a whole lot of these complementary pairs in a matrix. Of course in the example below we are driving two LEDs with two microcontroller pins....you could say why bother.

Well the next section is where we get to the guts of charlieplexing and how it makes an efficient use of a microcontrollers output pins.



step 4: Finally....a Charlieplex matrix

As mentioned in the introduction, charlieplexing is a handy way of driving lots of LEDs with only a few pins on a microcontroller. However in the previous pages we've not really saved any pins, driving two LEDs with two pins....big whoop!

Well we can extend the idea of complementary drive into a charlieplex matrix. The diagram below shows the minimum charlieplex matrix consisting of three resistors and six LEDs and using only three microcontroller pins. Now do you see how handy this method is? If you wanted to drive six LEDs in the normal way....you'd need six microcontroller pins.

In fact with N pins of a microcontroller you can potentially drive $N * (N - 1)$ LEDs.

For 3 pins this is $3 * (3 - 1) = 3 * 2 = 6$ LEDs.

Things stack up quickly with more pins. With 6 pins you can drive $6 * (6 - 1) = 6 * 5 = 30$ LEDs....wow!

Now to the charlieplexing bit.

Look at the diagram below. We have three complementary pairs, one pair between a each combination of micro output pins. One pair between A-B, one pair between B-C and one pair between A-C.

If you disconnected pin C for now we'd have the same situation as before. With 5V on pin A and 0V on pin B, LED1 will glow, LED2 is reverse biased and will not conduct current. With 5V on pin B and 0V on pin A LED1 will glow.

This follows for the other micro pins.

If we disconnected pin B and set pin A to 5V and pin C to 0V then LED5 would glow. Reversing so that pin A is 0V and pin C is 5V then LED6 would glow. Same for the complementary pair between pins B-C.

Hang on, I hear you say. Lets look at the second case a bit more closely.

We have 5V on pin A and 0V on pin C. We've disconnected pin B (the middle one).

OK, so a current flows through LED5, current isn't flowing through LED6 because it is reverse biased (and so are LED2 and LED4)....but there is also a path for the current to take from pin A, through LED1 and LED3 isn't there? Why are these LEDs not glowing as well.

Here is the heart of the charlieplexing scheme. Indeed there is a current flowing both LED1 and LED3, however the voltage across the both of these combined is only going to be equal to the voltage across LED5. Typically they would have half the voltage across them that LED5 has. So if we have 1.9V across LED5, then only 0.95V will be across LED1 and 0.95V across LED3.

From the I_f/V_f curve mentioned at the beginning of this article we can see that the current at this half voltage is much much lower than 20mA.....and those LEDs will not glow visibly.

This is known as current stealing.

Thus most of the current will flow though the LED we want, the most direct path through the least number of LEDs (ie one LED), rather than any series combination of LEDs.

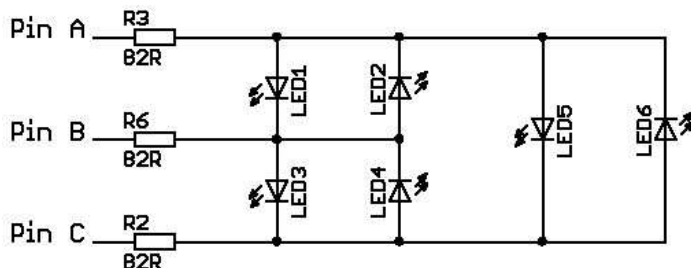
If you looked at the current flow for any combination of putting 5V and 0V on any two drive pins of the charlieplex matrix, you'll see the same thing. Only one LED will glow at a time.

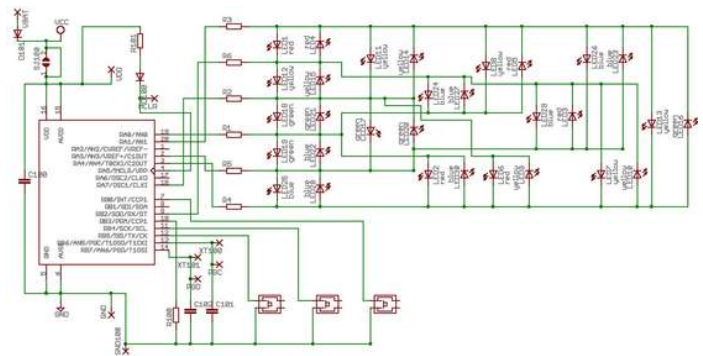
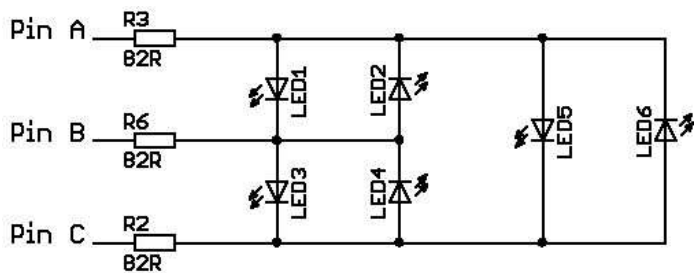
As an exercise, look at the first situation. 5V on pin A and 0V on pin B, disconnect pin C. LED1 is the shortest route for the current to take, and LED 1 will glow. A small current will also pass through LED5, then back up LED4 to pin B.....but again, these two LEDs in series will not be able to syphon enough current compared to LED 1 to glow brightly.

Thus the power of charlieplexing is realised. See the second diagram which is the schematic for my Microdot watch.....30 LEDs, with only 6 pins. My Minidot 2 clock is basically an expanded version of the Microdot....same 30 LEDs arranged in an array.

To make a pattern in the array, each LED to be illuminated is briefly switched on, then the micro moves to the next. If it is scheduled to be illuminated it is switched on again for a brief time. By quickly scanning through the LEDs fast enough a principle called 'persistence of vision' will allow an array of LEDs to show a static pattern. The Minidot 2 article has a bit of an explanation on this principle.

But wait.....I've seemingly glossed over a bit in the description above. What's this 'disconnect pin B', 'disconnect pin C' business. Next section please.





step 5: Tri-states (not tricycles)

In the previous step we mentioned a microcontroller can be programmed to output a 5V voltage or a 0V voltage. To make the charlieplex matrix work, we select two pins in the matrix, and disconnect any other pins.

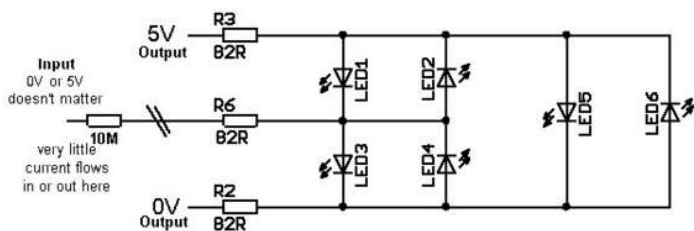
Of course manually disconnecting the pins is a bit difficult to do, particularly if we are scanning things very quickly to use the persistence of vision effect to show a pattern. However a microcontroller output pins can also be programmed to be input pins as well.

When a micro pin is programmed to be an input, it goes into what is called 'high-impedence' or 'tri-state'. That is, it presents a very high resistance (of the order of megaohms, or millions of ohms) to the pin.

If there is a very high resistance (see diagram) then we can essentially regard the pin as being disconnected, and so the charlieplex scheme works.

The second diagram shows the matrix pins for each combination possible to illuminate each of the 6 LEDs in our example. Typically a tri-state is denoted by an 'X', 5V is shown as a '1' (for logical 1) and 0V as a '0'. In the micro firmware for a '0' or '1' you'd program the pins to be an output and it's state is well defined. For tri-state you program it to be an input, and because it's an input we don't actually know what the state may be....hence the 'X' for unknown.

Although we might allocate a pin to be tri-state or an input, we don't need to read it. We just take advantage of the fact an input pin on a microcontroller is high impedance.



LED	Pin A	Pin B	Pin C
LED1	1	0	X
LED2	0	1	X
LED3	X	1	0
LED4	X	0	1
LED5	1	X	0
LED6	0	X	1

step 6: Some Practical matters

The magic of charlieplexing relies on the fact the individual voltage presented across multiple LEDs in series will always be less than that across one single LED when the single LED is in parallel with the series combination. If the voltage is less, then the current is less, and hopefully the current in the series combination will be so low that the LED will not light.

This isn't always the case however.

Lets say you had two red LEDs with a typical forward voltage of 1.9V in your matrix and a blue LED with a forward voltage of 3.5V (say LED1=red, LED3=red, LED5=blue in our 6 LED example). If you lit up the blue LED, you would end up with $3.5/2 = 1.75V$ for each of the red LEDs. This may be very close to the dim operating area of the LED. You might find the red LEDs will glow dimly when the blue is illuminated.

It is a good idea therefore to make sure the forward voltage of any different coloured LEDs in your matrix are roughly the same at the operating current, or else use the same coloured LEDs in a matrix.

In my Microdot/Minidot projects I didnt have to worry about this, I used high efficiency blue/green SMD LEDs which fortunately have much the same forward voltage as the reds/yellows. However if I implemented the same thing with 5mm LEDs the result would have more problematical. In this case I would have implemented a blue/green charlieplex matrix and a red/yellow matix separately. I'd have needed to use more pins....but there you go.

Another issue is to look at your current draw from the micro and how bright you want the LED. If you have a big matrix, and are rapidly scanning it, then each LED is on for only a brief time. This it will appear relatively dim compared to a static display. You can cheat by increasing the current through the LED by reducing the current limiting resistors, but only to a point. If you draw too much current from the micro for too long you'll damage the output pins.

If you have a slowly moving matrix, say a status or cyclon display, you could keep the current down to a safe level but still have a bright LED display because each LED is on for a longer time, possibly static (in the case of a status indicator).

Some advantages of charlieplexing:

- uses only a few pins on a microcontroller to control many LEDs
- reduces component count as you don't need lots of driver chips/resistors etc

Some disadvantages:

- your micro firmware will need to handle setting both voltage state and input/output state of the pins
- need to be careful with mixing different colours

<http://www.instructables.com/id/Charlieplexing-LEDs--The-theory/>

- PCB layout is difficult, because the LED matrix is more complex.

step 7: References

There are lots of references about charlieplexing on the web.
In addition to the links at the front of the article, some of them are:

The original article from Maxim, this has a lot to say about driving 7 segment displays which is also possible.
http://www.maxim-ic.com/appnotes.cfm/appnote_number/1880

A wiki entry
<http://en.wikipedia.org/wiki/Charlieplexing>

Related Instructables



3D LED Charlieplex Cube from Christmas Tree Lights by rgbphil



A CharliePlexed RGB LED Dice by cedtlab



5x4 LED display matrix using a Basic Stamp 2 (bs2) and Charlieplexing by barney_1



DIY Electronic Birthday Blowout Candles by cedtlab



Minidot 2 - The holoclock by rgbphil



GuGaplexed Valentine LED Heart by cedtlab



LED Hanukkah Menorah by barney_1



Interactive Ambient Light by Hazard



Comments

50 comments [Add Comment](#)

[view all 58 comments](#)



Carlos Marmo says:
Wonderful Work!
Congratulations!

Oct 30, 2008. 2:35 AM [REPLY](#)



electric_destruction says:
NICE!!! i havent been able to get an account 4 a while, but i hav been reading instructables for monthes. urs really helped me out with a problem i had 2 solve. AWSOME!!!

Aug 26, 2008. 6:38 AM [REPLY](#)



McLaren says:
Please forgive the typo' in the previous post. Obviously I meant to say; **This improves overall brightness by increasing duty cycle.**
I suspect there's a few chaps scratching their heads trying to figure out how to drive that 6 pin 30 LED matrix example in my previous post so I've attached a driver example in C.

Jun 15, 2008. 7:25 AM [REPLY](#)

Basically, it uses a 6 element array, one element for each column, with the least significant five bits in each element corresponding to the five LEDs in each column. Your *main* program simply sets or clears those bits to turn the corresponding LEDs on or off. The ISR driver uses periodic timer 2 interrupts and lights the LEDs in a single column during each interrupt interval so it takes six interrupt intervals to refresh the entire matrix.

The driver may not seem very intuitive at first but if you look at the LEDs on the RB5 row in the drawing (link) in my previous post and think of RB5 as a "floating" row, the method may make more sense.

Have fun. Regards, Mike

K8LH Charlieplexed Driver Example



rgbphil says:

Jun 15, 2008. 10:38 PM **REPLY**

Hi Mike,

I've had a very quick look at the code, can you answer a couple of questions though.

If you are only using 6 interrupt intervals, then how can you provide an arbitrary pattern for 30 LEDs without illuminating several LEDs at once?

This might be your purpose however, so please excuse me if that is the case. From what I understand you are doing a bargraph/bardot display. In that case, this is a reasonable method for such displays.

For the minidot/microdot clocks however I wanted an arbitrary pattern for the 30LEDs (see pics on these instructables).....not sure that this would help in that case. If you have a look at the minidot/microdot code, I've had to have a lookup table for each LED and put the tris/port values out for each possible combination.

I suppose the 'all possible combination' method is the generic method for pur charlieplex arrays, and this method is better suited as you've noted for seven segment displays or bar graphs where either only one LED is illuminated, or an ordered number of LEDs (ie a bar) is illuminated.

I might be missing something, if you can do another more efficient algorithm for arbitrary patterns please post it.

Again, I mentioned I've only had a quick look. On any particular iteration through the 6 interrupt cycles, do you have several LEDs on at once? If so, then you might have a problem with varying brightness if say on cycle one you want one LED, cycle two four LEDs etc because the cumulative forward voltage of several LEDs being illuminated will change the current.

Thanks for adding to the instructable though, I think it illustrates the different cases for charlieplex arrays.

Phil



rgbphil says:

Jun 15, 2008. 11:06 PM **REPLY**

>>>>RETRACTION<<<<<<<

Sorry!!!! I must do a bit of back paddling here. I didn't look at the solution you've come up with properly.

I didn't refer to the below circuit diagram when making the above comments.

I see now how you can have several LEDs on at once and still have an arbitrary pattern.

Lets see if I can explain properly.

First, because the diagram has no LEDs numbers lets assume a numbering convention where L11 means LED at row 1, column 1, top left hand corner and L61 means LED at row 6, column 1 (bottom right hand corner. Of course LED11, LED22, LED33 etc don't actually exist in the matrix.

You can set an arbitrary pattern for column 1, eg L21, L31, L41 by setting tris/port values tris=xx110000 and port=xxx10001 (where x means don't care).

This sets RB0 to high to turn on the column driver, and RB1,2,3 to LOW to turn on LEDs L21,L31 and L41. RB4 and RB5 are tri-state.

This would be the first interrupt cycle.

Next pattern for column 2 eg L21, L23, L26 by setting tris=xx010100 and port=xx0x0x10

and so on. You can see that for each column in your arrangement there will be the same number of zeros in the port setting as the number of LEDs in that column you want to illuminate, plus a 'one' to set the output column driver. In the tris, there will be the number of desired LEDs+1 zeros to enable the output lines (all others to tri-state).

Apologies for jumping the gun before....this is a great method for reducing the number of interrupt cycles to control the array. With the extra time saved, PWM can be used to do brightness control.

My original concern was that the current sinking capability of the micro wasn't enough to handle several LEDs at one, however I can see now that each current sink line, only sinks one LED worth of current, and the transistors source the needed current.

Hats off....thanks for helping the charlieplex community out. I can see the validity of this method now. You do need extra components....but that could be handled by a single transistor array chip (eg ULN 2800 for PNP drivers, ULN2803 of NPNs if you arrange it the other way).

When I extract my finger, I'll be using this method on my little LED cube. I was worried about getting PWM brightness control for a 5x5x6=125 LED array.

Phil



McLaren says:

Jun 16, 2008. 4:23 AM **REPLY**

You've almost got it Phil. The PORT reg' can only have a single '1' bit at any time (xx100000, xx010000, xx001000, xx000100, xx000010, or xx000001) to turn on a single column driver transistor. That's the purpose of the **colpos** ring-counter variable in the driver. We also use this '1' bit in **colpos** to determine when an individual led bit in our display array variable matches one of those invalid col/row positions (rb0/rb0, rb1/rb1, etc.) and needs to be "floated" onto the RB5 "float" row. If you physically slide those RB5 "float" row LEDs into the holes in the matrix above then your display array will match your LED numbering scheme.

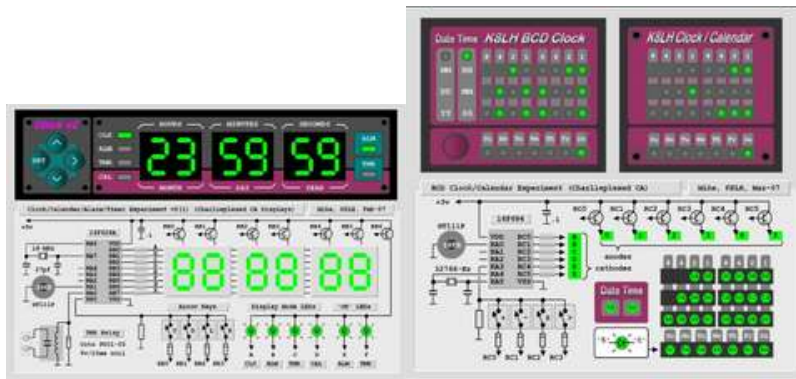
Yes, you can still PWM the LEDs with this method (with improved duty cycle).

Yes, you could use a UDN2981 sourcing driver IC for CA columns or a ULN2803/2804 sinking driver IC for CC columns but adding an IC almost defeats the purpose of Charlieplexing (grin). I guess for some reason I don't think adding 5 transistors is 'cheating' (grin).

BTW, during my Charlieplexing experiments several years ago I observed higher brightness when the PIC pins sink current to LED cathodes rather than source current to LED anodes and so that's why you'll notice common anode column drivers in most of my projects.

Have fun.

Mike, K8LH (Michigan, USA)



dforsyth says:
what simulator are the product pictures from?

Jun 19, 2008. 9:26 AM [REPLY](#)



McLaren says:
Those drawings are Microsoft Excel spreadsheets (using the drawing tools).
Mike

Jun 19, 2008. 10:38 AM [REPLY](#)



carlitoscr says:
Hi McLaren, Very nice drawings, why don't you create an instructable on how to make them?

Aug 12, 2008. 2:26 PM [REPLY](#)



McLaren says:
Actually now that I think about it, I don't think you could use ULN 2803 or UDN2981 driver ICs because they don't follow the Charlieplexing active-low or active-high rules.
Sorry... Mike

Jun 16, 2008. 5:09 AM [REPLY](#)



Myself says:
Looking at the 30-LED array on the Microdot project really helped drive the point home. It'd be cool to see a 4-pin (12-LED) and 5-pin (20-LED, if I did my math right?) layout too, just to watch it all take shape.

May 23, 2007. 2:48 PM [REPLY](#)

Neat stuff!

One problem with multiplexed LED drives is that if there's any vibration in the system, the image jiggles and looks horrible. Car displays, moving-message signs, even clock radios if I lean my head against the doorframe while the washing machine's running, they all fall apart. I guess there's no way around this? Initially I figured you could put capacitors on each LED to smooth out the pulse, but then I realized that the scheme relies on each pin changing state quickly, and capacitors would interfere with that.

On that note, then, would very long wires between the micro and the array tend to cause problems because of their intrinsic capacitance?



carlitoscr says:
Hi, in this kind of applications the only way that I know to mitigate that problem is using a high scanning frequency. The higher the frequency the lesser the susceptibility to break the image by vibration and sudden movements.

Aug 11, 2008. 7:27 PM [REPLY](#)



rgbphil says:
You're right, putting capacitance on the lines would defeat the purpose.

May 23, 2007. 10:56 PM [REPLY](#)

The solutions to jiggles is to have a faster scan rate than the vibration frequency, have a persistence effect on the 'pixels' like a CRT television or some sort of static state of the pixels...like those new fangled E-paper dots. For LEDs the latter two aren't an option, so you need to scan faster. It's a common problem to any sort of multiplexing, whether charlieplexing, seven segment or matrix.

One way to do this is to split a large array of LEDs into smaller array sub-units. So if you wanted 60 LEDs, you might use three charlieplex arrays of 20 LEDs each. This would require $3 \times 5 = 15$ pins (5 pins gives 20 LEDs) instead of 8 pins so there is a tradeoff for pins v refresh rate. I've found that about 40-60 LEDs off a 16F88 PIC is about the practical limit with regards to refresh rate. The nebula confinement chamber has 66 LEDs, 44 blue/green and 22 red ones arranged in two charlieplex grids.

Very long wires would indeed play havoc, but in practice you wouldn't remote the power source from the LED array. And in any case, being limited to a few tens of mA from the micro wouldn't raise that much of an issue, unless you're talking about 20m of wire and scanning at 10kHz or more! The big issue with capacitance on the wires is when you move from a logic '1' or '0' to a tri-state value, the current wouldn't be sunk or sourced away fast enough and would need to discharge through another path, leading to a blurry display.

I'm currently trying to work out how to buffer a tri-state signal to make a high current charlieplex array to use 100mA superflux LEDs, low current signals would drive a big array so wirelength will be less of an issue.....stay tuned.



dleake says:

I have a question about the PWM. Your code is basically:
foreach column / foreach PWM period / set the outputs
What difference(s) would it make if instead it was:
foreach PWM period / foreach column / set the outputs

Aug 5, 2008. 10:02 AM [REPLY](#)



alain91 says:

Excellent article. Thanks a lot !

Jul 7, 2008. 1:55 PM [REPLY](#)



rgbphil says:

Well here's a pic of my setup. The column is a bit over a meter long, has 30 green superflux LEDs running at about 50-60mA. Currently running a stock standard cylon eye pattern thanks to Mikes PWM program, with a few modifications to make it work on a 6 channel charliplex rig. Incidentally....had a bit of a brainwave, there were some issues related to the need for the input voltage on the high side to be higher than the combined Vces of the driver transistors plus LED plus limiting resistor...making a full current drive of the LEDs difficult because of the tight fit (in voltage). Also made using a Blue LED impossible. However I reckon that using an optocoupler might do the trick. One per high side driver. They are cheap, can handle high currents, and either side of the opto can be at different voltages, meaning I can run my micro at 5V, and raise the driver side to something comfortable for high current LEDs at 6-7V. Even higher if I went crazy and put several LEDs in series per matrix cell. Will give it a go and report.

Jun 23, 2008. 6:07 AM [REPLY](#)



McLaren says:

Very nice. Bravo Phil...

Jun 23, 2008. 8:04 PM [REPLY](#)



McLaren says:

Just wanted to mention an interesting development as I played with a new PWM-32 version of the 12F683 program.

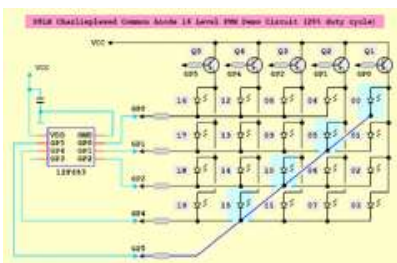
Jul 1, 2008. 7:12 PM [REPLY](#)

The animation and fading effects didn't seem as smooth as they should be until I added a line to sync' the led variable updates to the 16 msec frame rate of the matrix driver. **WOW, what a difference!**

The Charlieplexed PWM-32 video simply doesn't show how nice the effect is.

Anyway, here's the latest 12F683 PWM-32 Demo program (written using the Lite/Free version of BoostC) for those who want to play.

Have fun. Mike



rgbphil says:

Had a quick look at the program, if I'm reading it right you're waiting till the end of a refresh period to update the led brightness array?

Jul 3, 2008. 8:27 PM [REPLY](#)

One enhancement you might want to try is to have two buffers. One has the current led array being displayed the other has the one that your main program updates. Then the interrupt program switches between the two and the end of each refresh period automatically.

You'd need to write a little routine to change the LED brightness in the appropriate buffer, and perhaps a small critical block right where the value is written to the buffer to make sure the interrupt doesn't switch buffers in the middle of a change.

Phil



rgbphil says:

Thanks Mike....will look into the new program you've put up, may take a bit of time, have some work to finish first.

Jul 2, 2008. 8:48 PM [REPLY](#)



rgbphil says:

Hi All,

Much thanks to Mike for coming up with an improvement to the charlieplexing driver circuit. After many misunderstandings on my part, I can see how he's done it.

Jun 19, 2008. 9:57 PM [REPLY](#)

This comment is to consolidate some of the commentary below. I'll update this instructable when I've made some circuits of my own.

From Mikes diagram in the post immediately below, we can see a set of five NPN transistors. Note that their collectors are connected to the Vdd line, and their emitters are each connected to a row of LED anodes.

<http://www.instructables.com/files/deriv/FX7WSQ2/FHM8R2A6/FX7WSQ2FHM8R2A6.MEDIUM.jpg>

This is an unusual approach (certainly fooled me), normally you'd see NPN transistors living at the bottom of a circuit diagram, with their emitters tied together.

However the transistor will still turn on, as long as sufficient current flows from the base to the emitter, or in other words if the base voltage is raised above the emitter voltage.

This will be the case in this diagram.

For example to turn on LEDs L00, L02 and L03 you would set GP0=HIGH, GP2=LOW, GP4=LOW and GP5=LOW...all other ports to tristate.

Q1 turns on, because the base voltage is higher than the emitter voltage and current will thus flow into the base. The transistor will then supply current to the LEDs, which are individually sunk by GP2, GP4 and GP5.

We won't have a problem with other LEDs we don't want illuminated turning on, because we've only raised GP0 to high, and because the cathodes of L16, L12, L08 and L04 are high they are effectively reversed biased, even if GP2, GP4 and GP5 are pulled low.

Using this technique, Mike has shown that in a single pass through an loop (usually implemented via an interrupt) he can turn on several LEDs. Thus you only need as many loops as there are column drivers. This saves enormous amounts of processor time, which can be put to better use in executing a PWM routine to provide both arbitrary patterns, as well as arbitrary brightness levels to each LED.

In case you are wondering if this is not still 'charlieplexing', it is. The reason is that we are still forward biasing the LEDs we want on, and reverse biasing the LEDs we don't. Effectively Mike has an array arranged so that one LED is forward biased with several LEDs reversed biased....different from my arrangement where there was only one of each, and we relied on the cumulative forward biasing of unwanted LED chains to be higher than the one we wanted to avoid illuminating the undesired LEDs (ie current stealing).

The one thing to note, is that the base resistors will need to be higher in value than the charlieplex line resistors or else too much current will be drawn. For example, if L00 is on, and its forward voltage is for example 2.1V, current is 20mA, then assuming a resistance of say 100ohm for the charlieplex resistor the emitter voltage will be 4.1V ($2.1 + 0.02 \times 100$), add the base emitter voltage of 0.6V (typical) then we end up with a base voltage of 4.7V. With a GP0 voltage of 5V, the voltage across the base resistor will be 0.3V, assuming a current gain of say 80, then we need a current through this base resistor of $0.02/80 = 0.25\text{mA}$ or a resistor value of $0.3/0.00025 = 1.2\text{k}$all typical junkbox values. If we had lower resistor values, then we'd get much higher base currents. You also need to use transistors capable of a low Vce.

As mentioned in the comments below, I'll update this instructable after trying the method out with my LED cube.

Again, thanks to Mike for joining in the Instructables spirit of community and sharing his technique.

Phil

PS: after going through the math above, I think Mike has come up with a way of doing the holy grail of charlieplex arrays, making the charlieplexing system work with high current LEDs. Instead of sinking the LED cathodes directly to the microcontroller port pins, instead attach them to PNP transistors, with their emitters connected to the cathodes of the LEDs, and their collectors connected to ground (via a limiting resistor), their bases would be connected to the micro. Thus setting the port pins of desired LED ROWS to ground would sink that row, setting the port pin of the desired column to HIGH would source that column. Any undesired LEDs would not be illuminated because the base voltage would be higher than their emitter and the base/emitter junction of the row driver would be reverse biased.

woowhooooo!!!! I've very excited about this possibility. Might even give that a go first before the LED cube!



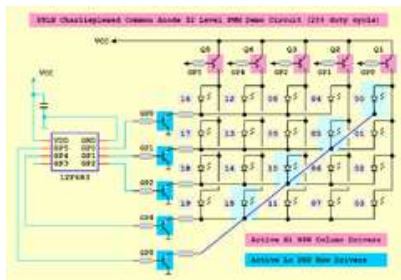
McLaren says:

Active low PNP drivers on the cathode rows should work.

Jun 22, 2008. 1:14 PM [REPLY](#)

I added a single PNP row driver on the GP0 row to compare brightness and the four LEDs on that row were noticeably brighter than the other 16 LEDs (all were set to pwm level 1).

Mike



rgbphil says:

Jun 22, 2008. 5:17 PM [REPLY](#)

Hi,

I've actually made up a little test circuit with high side NPN drivers and low side PNP drivers...and it works, with some limitations you may not have encountered using low current LEDs.

Firstly I wanted to use 100mA superflux spider LEDs, as I was implementing a 30 LED charlieplex matrix, this meant the column driver needed to be able to source upto 500mA of current. An improbably high gain would be needed to get the base current under 20mA....too much for the microcontroller to bear. The BC337 datasheet I used states a $V_{ce(sat)}$ at 500mA, but at $I_b=50mA$too much. So I implemented a darlington driver for the high side column drivers.

As the low side would only need to sink at most 100mA....I could get away with a single transistor as the base current would be under 20mA.

Now this results in two effects.

I used BC337 and BC327 transistors which have a $V_{be(on)}$ voltage of 1.2V and a $V_{ce(sat)}$ of 0.7 at 500mA.

Firstly the V_{ce} of the column driver is around 1.2V because of the combined effect of $V_{be(on)}$ and $V_{ce(sat)}$. I say around, because the effective V_{ce} of the darlington pair will vary depending on the current drawn.

So subtracting $V_{ce(sat)}$ for the darlington, and $V_{ce(sat)}$ for the PNP low side driver I get $5-1.2-0.7=3.1V$. This is enough to put in red LEDs with a forward voltage of 2.1V plus a limiting resistor....however I wanted to use green, with a forward voltage of 3.2V at 100mA.

So for my circuit I omitted the limiting resistor and am operating the LED at a lower part of the response curve. I'm getting about 70mA. It's a little like going trail bike riding without a helmet.

Note again the about. When more LEDs are illuminated per column, the current increases, and thus the $V_{ce(sat)}$ increases, dropping the available voltage to the LEDs, thus lowering the current. So one LED ON will have a higher current than several LEDs ON (per LED), ie a non linear effect....which may make keeping a consistant brightness difficult.

For a varying pattern this might not be too much of an issue, but is a measurable effect (using an ammeter I get from 50mA to 70mA difference depending on how many LEDs are on.

So the high side NPN low side PNP driver scheme does work for high currents, however you do need to pick your transistors very carefully. And in either case you will still have the problem of different currents flowing due to varying column voltages.

Note for low current LEDs this will be less of a problem.

I'm beginning to think that mosfets will be needed to implement more controlled high current active charlieplex matrices.

Will post again when I've implemented a nice PWM algorithm.

Phil



McLaren says:

Jun 19, 2008. 8:33 PM [REPLY](#)

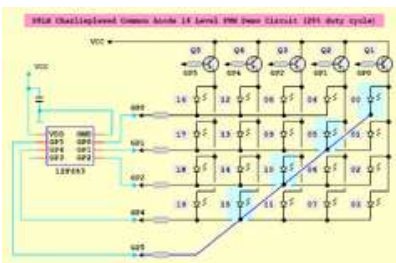
If anyone is interested, here's the BoostC source code and schematic for the **12F683 Charlieplexed PWM20 Demo**.

It's 20 Charlieplexed LEDs with 16 levels of PWM brightness control per LED.

You're not going to believe how easy it is to control these LEDs. All of the real work is being done in the ISR.

BoostC PWM20 Source Code

Have fun. Regards, Mike





McLaren says:

Jun 18, 2008. 11:06 PM [REPLY](#)

Never thought of applying pwm to Charlieplexed displays for brightness control or fading effects until reading your instructables Phil. Thank you.

I just finished testing 16 level pwm on a little Charlieplexed 5 pin 20 LED project board (an 8 MHz 12F683 with my standard transistor column drivers) and I'm really "geeked". It works great. Let me know if you'd like to see a picture.

I'm using 16 pwm 'steps' per LED at 125 usec interrupt intervals per 'step' (that's also per column, btw) for a whopping 20 msec 100 Hz refresh rate. The ISR is using 51 to 78 cycles per 125 usec interrupt interval which would be about 12.5% ISR overhead with a 20 MHz clock. It's also got the simplest interface to Main that I could think of (none of those bitmask constant arrays);

```
unsigned char led(20);
```

```
led(0) = on; . . . // "on" is defined as 15
```

```
led(1) = off; . . . // "off" is defined as 0
```

```
led(2) = n; . . . // "n" can be any number between 0 and 15
```

Scaling it up to 30 LEDs (6 columns and 5 rows) shouldn't be a problem though I'll probably use 100 usec interrupt intervals and a 104 Hz refresh rate. The ISR overhead will also increase to around 58.85 cycles (17 usecs per 100 usec interval with a 20 MHz clock -- 17% ISR overhead).

Thanks for the inspiration...

Regards, Mike



McLaren says:

Jun 19, 2008. 7:07 AM [REPLY](#)

Some additional analysis and a not-so-good picture.

Your method of lighting only 1 of 30 LEDs at a time with 8 soft' pwm steps produces LED duty cycles of 0.42% minimum (duty cycle = 1) to 3.33% maximum (duty cycle = 8).

Driving 5 of 30 LEDs at a time in a Charlieplexed column and row matrix with transistor column drivers makes a huge difference and produces LED duty cycles of 1.04% minimum (duty cycle = 1) to 16.64% maximum (duty cycle = 15). You can also achieve much higher refresh rates with the extra time available.

You really should consider this method in a future project. With nearly 5 times increase in duty cycle you should notice a big difference. And all this extra performance only costs about 60 cents for the 6 transistors (grin).

The picture came out awful and just doesn't show the smooth brightness transitions that I'm seeing. BTW, those are green LEDs. and the pwm weighting that I'm using is (lowest to brightest) 1,2,3,4,5,6,7,9,12, and 15 on both sides.

Regards, Mike



rgbphil says:

Jun 19, 2008. 1:06 AM [REPLY](#)

Hey its great your got something to work so well.

Did you get a chance to have a look at the comments below...still wondering how you've avoided getting multiple LEDs to light up.

Just had a thought....why not put up an instructable with cct diagram and code? Great way to get fame and fortune (....well maybe not fortune).

Phil



McLaren says:

Jun 19, 2008. 10:43 AM

(removed by community request)



rgbphil says:

Jun 19, 2008. 9:17 PM [REPLY](#)

ahhhhhhhh.....much apologies again. I was looking at the placement of the transistors and just assuming they were PNP because traditionally NPNs are put down the bottom of a diagram.....doh

Alright, now I see how everything works. If the base voltage (say 5V-resistor voltage) is higher than the emitter voltage (forward voltage of the LED) then the transistor will turn on, with an active high on the charlieplex line as required.

OK.....

Well if you don't mind, I'll try to get around to amending this instructable with the method you've come up with. Hopefully by first trying it out on my LED cube. For the mean time, I'll post a new reply at the top of this chain to clarify how things are working for those having trouble following the thread.

This also explains why an ULN2803 or similar transistor array chip wont work, because their emitters are all tied to the same line. You could still

use a transistor array chip, but only one that has separate transistors not grouped together.

Again...very nice work.

Phil



McLaren says:

Jun 19, 2008. 9:54 PM [REPLY](#)

Whew! It almost seemed (to me) like you weren't interested enough to pay attention. I'm happy you finally "get it".

Now perhaps we can kill the myth about only being able to light one Charlieplexed LED at a time and let people know they can get some real performance out of these things.

The other challenge will be convincing authors and authorities like you to start thinking of and describing Charlieplexed displays in terms of the column and row matrices they really are. Once you understand you'll find yourself writing much simpler and higher performance code to drive these displays.

Take care. Kind regards, Mike



McLaren says:

Jun 16, 2008. 7:11 AM [REPLY](#)

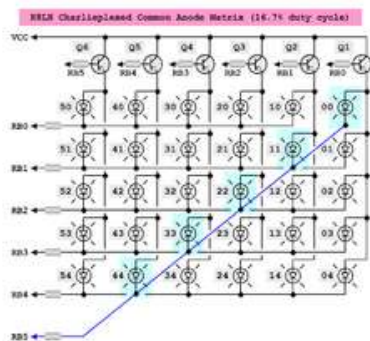
Not an easy concept to get your head around. And as I mentioned earlier, I'm not nearly as good at explaining concepts as Phil is. Still, I feel the "float" mechanism needs additional explanation so here goes.

I've made another drawing (below) and I've numbered the LEDs as Phil suggested as a 2 digit column and row number (which also closely matches our display array variable). I've also moved all of the LEDs in the 6th row up into the holes that were in the previous matrix drawing. Those holes were showing the invalid Charlieplexed column/row combinations (RB0/RB0, RB1/RB1, etc.).

The "float" mechanism allows us to use consistent bit patterns in our LED array variable. Each `led()` array element 0..5 corresponds directly to the column pin (RB0..RB5) and the b4..b0 bits in each array element correspond directly to the row pin (RB4..RB0). Of course this means that LED bits at `led(0).bit0`, `led(1).bit1`, `led(2).bit2`, `led(3).bit3`, and `led(4).bit4` are invalid because the column and row pins associated with these bits is the same pin. That's why we "float" those row bits in the driver onto the RB5 "float" pin and why those "float" LEDs occupy the spots they do in the matrix.

I hope the new drawing and the driver example in my previous post makes more sense now.

Regards, Mike



rgbphil says:

Jun 16, 2008. 5:42 PM [REPLY](#)

mhm.....looking at it again, now not sure. I think we've both make a mistake in the polarity of the signal needed to enable the column driver. The PNP current sourcing transistors will need their bases pulled down to LOW to turn on.

Lets take a simple example, we want to illuminate L00 and L02 in one pass.

We set RB0 LOW to activate the column driver Q1, pulling down the base.

We also set RB5 LOW (your floating column, now nicely explained) to illuminate L00 and RB2 to illuminate L02.

All well and good, however won't Q3 also be activated by RB2, thereby illuminating L22 because RB5 is LOW?

Another example, we want to illuminate L01 and L02 so we set RB0 LOW to activate Q1, and RB1 and RB2 to pull down the cathodes on L01 and L02. However this also activates columns Q2, Q3, meaning LEDs L12 and L21 are illuminated through cathodes on RB1 and RB2 respectively?

I'm probably missing something here. Can you give an example as I've done above, explicitly saying in pass x, we want to illuminate LEDx,y,z and hence we need tris/port values a,b please.

I can see how this arrangement will work, however we'd need to invert the signals to the driver transistors so you can send an active HIGH to the column driver, this would then not provide a current sink path to LEDs we DON'T want to illuminate....more parts, but less time to scan the array. Maybe a nice inverting buffer chip instead of two transistors per column.

Thanks
Phil



rgbphil says:

Jun 17, 2008. 7:19 PM [REPLY](#)

actually it would have to be an open collector non inverting buffer chip



McLaren says:

Jun 15, 2008. 4:45 AM [REPLY](#)

I wish I could explain things as well as you Phil. Bravo on a fine set of instructables.

I did want to let you know I figured out how to light multiple Charlieplexed LEDs at the same time to even brightness several years ago. This as you know improves overall brightness by reducing duty cycle.

The method is easier to understand if you draw the matrix in a more traditional column and row format (see link). Use NPN column drivers for common anode columns or use PNP column drivers for common cathode columns. Seems backwards I know but it follows the Charlieplexing rules. That is, an NPN sourcing driver would be "active high" while a PNP sinking driver would be "active low"

In the example 6 pin 30 LED common anode matrix (see link), only one pin will ever be driven high at any time to turn on one of the NPN column drivers. The other 5 pins will be driven low to turn on a column LED or set to hi-z (input) to turn off a column LED. The NPN drivers easily source the cumulative sinking current from the other five I/O pins and you get the same even brightness when lighting a single LED or all five LEDs in a column.

Have fun. Kind regards, Mike McLaren, K8LH (Michigan, USA)

Charlieplexed 6 pin 30 LEDs



alexfhalford says:

Jun 10, 2008. 2:41 PM [REPLY](#)

That's a really well explained instructable.

Favourited.

Alex Halford



Learndy says:

Apr 25, 2008. 12:05 AM [REPLY](#)

Okay. I am currently building a Microchip PIC charlieplexer driving 8 LEDs via 4 outputs. Prototype plugged together on breadboard, program ran the first time yesterday evening. Seems to work as expected: LEDs are semi-bright.

My PIC can drive 20 (25?) mA per pin. I would like to connect bright LEDs drawing maximally 30 mA or even up to 100 mA at low duty cycle. I would like to give them as high current as possible for maximum brightness.

Is there any simple way to boost tri-state output? I fear that I will have to switch to a PIC in 14 pin instead of 8 pin package. Oh, I forgot: If you look at the lights several of them look bright at the same time. So switching on only one of them is no alternative.

virtuPIC

--

Airspace V - international hangar flying!

<http://www.airspace-v.com>



rgbphil says:

Apr 25, 2008. 3:06 AM [REPLY](#)

Hi,

Congratulations on doing a charlieplex circuit...would be keen to see it on instructables when its complete.

As for your problem.....it's one I've been trying to solve for a while, no real solution right now, but some ideas.

Initially I tried simulating a circuit in LTSpice using bipolar transistors with NPN/PNP in a push/pull configuration, but didn't have any luck getting the tri-state operation.

I've have some better luck with using P-type and N-type MOSFETS, though I still need to work out the proper way to bias the mosfets properly so that either can be 'ON' depending on the microcontroller output signal, and both 'OFF' when the output is tri-state.

Basically I suggest you replicate the output circuit of the microcontroller, which can be found from the datasheet.

Alternately, you could use bipolar transistors in a push-pull configuration, but have the base of the transistors driven two pins on the micro. Of course that means that you need two pins per charlieplex line....which sort of defeats the purpose.

Note in the instructable the need to ensure that the forward voltage is reasonably matched for all the LEDs you are using in the matrix. This may be the cause for you having some bright/dark. Another cause may be a wiring fault or check your code to make sure that all LEDs are getting equal time....try to make sure the 'OFF' LEDs get as much time as the 'ON' LEDs. I made a mistake once where I had my multiplexing code just processing the 'ON' LEDs per refresh cycle, and OFF LEDs getting no cycles, leading to brightness changing depending on the number of LEDs being lit.



Learndy says:

Apr 25, 2008. 5:15 AM [REPLY](#)

Well, the point is that for only 8 LEDs using two transistors plus respective resistors per LED is much more complicated and expensive than using a PIC with more I/O-pins.

In the current implementation I could use an 10F200 PIC in 8-pin package with 5 output pins at EUR 0.70 at digikey.de. An upgrade to 16F54 would bring a 20 pin package, 12 outputs at the same price! The complete circuit would be more expensive and would have more contacts to be soldered in the charlieplexing version.

Seems that I spare charlieplexing for higher number of LEDs / only one LED lit at a time. something like the LED cube with a light flying through it in a Lissajous-like way...

virtuPIC

--

Airspace V - international hangar flying!

<http://www.airspace-v.com>



rgbphil says:

Apr 25, 2008. 4:35 PM [REPLY](#)

Your right of course, I might have mentioned in the article the limitation on output current which seems to be your main concern.

Charlieplexing is best suited for a higher number of low current LEDs in a matrix as you've noted. One LED at a time is a fundamental limitation.

However I have used it in some applications where I have a micro (a 16F88) with a number of switch/sensor inputs but not quite enough outputs left for an indicator, or to reduce signal lines to a daughter board. For example I've used charlieplexing for only 4 LEDs (using three lines) simply because I couldn't fit more wires into a cable and there wasn't enough space for a bigger cable, and another situation where all but three pins on a micro were used for other purposes and I wanted 6 indicators but only three pins. Still it can be useful, say for example it would be trivial to make a 12 LED VU meter with an 8pin micro using 4 charlieplex lines, 2 pins power, 1 pin analog input and one for a switch/mode control.

Charlieplexing is just another trick in the book if you have such problems, usually not an issue for hobbyist applications.

If you have the pins available...then by all means use them, though you'd still have the current issue.

I'm not sure that doubling up on pins to increase current sourcing/sinking capability is a good idea, perhaps on a hobbyist project, however on a commercial product small imbalances due to component tolerances or timing tolerances might result in momentarily higher currents being sourced/sunk that would progressively degrade the reliability of the chip....so again you'd be stuck with extra components (and cost) to supply the current. It might be better to (in a professional or high reliability application) use two LEDs per indicator to increase the light output rather than a higher current LED (still using two pins though).

There are high current transistor arrays like the ULN2008 that can reduce the cost and complexity of adding many transistors you might want to look into. I've used them to source 100mA per LED. There are also high current I/O expanders that are controlled from only 2 lines in a I2C bus that can also be of use....though again at extra cost. You might try to get samples from manufacturers.

I'd like to work with anyone on the issue of a tri-state current expander for charlieplexing applications....so if you have any ideas, please post.



rgbphil says:

Apr 24, 2008. 5:54 PM [REPLY](#)

Thanks to all the people commenting and putting this article in the popular category. I'm glad it'd been very helpful in explaining charlieplexing to people.



pburgess says:

Apr 23, 2008. 10:51 PM [REPLY](#)

Thank you for explaining this with such clarity. I'd previously read a few explanations of Charlieplexing elsewhere, but it wasn't until this Instructable that the concept really "clicked."



burningsuntech says:

Apr 17, 2008. 8:25 PM [REPLY](#)

Wow!
Multiplexing on steroids. Great Instructable. Now a favorite!



GorillazMiko says:

Apr 17, 2008. 8:04 PM [REPLY](#)

Very cool, but extremely hard for me. :P

+1 rating.



Gjdj3 says:

Apr 17, 2008. 3:46 PM [REPLY](#)

That was a great article. I'm definitely going to have to try that. Another thing to add to the old project book. Nice Job!

+1



Macka says:

Feb 13, 2008. 3:46 AM [REPLY](#)

That was a fantastic article.
Handn't even heard about charlieplexing let alone know anything about it before i read this, now i feel like an expert :D well... maybe not quite, but you get the picture.

Very informative.



du says:

Feb 12, 2008. 8:52 AM [REPLY](#)

My officemate and I have been messing around with multiplexing schemes for a few weeks now. Neither of us is that knowledgeable, so this article was very interesting.

But we're both confused about the "tristate" thing. ...oh wait, while typing this comment I think I figured it out. To get + you write a HIGH to the pin in output mode, to get - you write a LOW. And then to null out the pin you put it in INPUT mode. OK, I get it.



Einsteins Circuitry says:

Jan 19, 2008. 7:35 PM [REPLY](#)

Wow. Very nice Instructable! One of the best I've seen. I've been looking for a good explanation for charlieplexing for a while now and just found yours. So far, your Instructable is the only charlieplexing explanation I've understood! *claps*:

Very nice! +1 and favorited



Ralph Corderoy says:

Jan 13, 2008. 9:06 AM [REPLY](#)

Thanks for the hard work that's gone into this article. Given the 3-pin 6-LED diagram, if A=1, B=0 *and* C=0 rather than tri-state, do LEDs 1 and 5 light or do the resistors being set for one LED mean the voltage through each of the two LEDs is too low? If the latter and you always wanted to light two LEDs, never just one, then you could choose the resistors appropriately?



rgbphil says:

Jan 13, 2008. 2:01 PM [REPLY](#)

This is a real good question.

I did some rough spice analysis of this situation a while ago, though I'd need to do some more to properly answer it. I was more concerned with the situation where there was one current sink (eg C=0) and two current sources (A=1 & B=1). In this case I found the current through each LED was slightly reduced.....however the current through the sink resistor was slightly increased. This was because of two current sources combining at C. The individual LED current reducing was because as the sink resistor current increased, the voltage across it increased, and therefore that reduced the available voltage for the LED. Because of the non linear curve of the LED, this was not easy to calculate by hand.

In the case of two current sinks and one current source the same thing will probably apply though in reverse, the micro output will source slightly more current, but each input will sink slightly less.....

If you check my christmas tree LED article where I have a massive charlieplex array being driven by a 15 channel PWM signal, I assumed that momentarily I'd have large current sinks or sourcing on the micro, however because the signal is rapidly changing there'd be no problem with heating, particularly as I'd set the resistors to be about 20-30% less than the maximum. I've had another PWM driven LED array going continually for about a year now and haven't killed the micro (see my minidot project...it's the 'Super Computer Status Array'.

As you can see it's a complicated thing to figure out, and I only did a rough analysis. My big problem was getting a good spice model of the SMD LEDs I was using. I remember that there was a way of doing these calculations graphically using load lines from my uni days.....but my memory is a bit fuzzy.

I'd be keen to work with you or any others who'd like to comment to derive a general system of calculations for multiple current sources/sinks in a charlieplex array.

To answer to your question, probably the easiest/quickest way to do this would be to rig a breadboard prototype and measure it directly.

Both LEDs will light up, though slightly dimmer. You need to worry about the effect on the microcontroller max current source/sink specs.



nrocy says:

Oct 16, 2007. 5:57 AM [REPLY](#)

Thanks for this, great explanation :)

I'm currently exploring my options for controlling a large amount of RGB LEDs from a microcontroller (4x4x4 cube of RGB LEDs).

Any suggestions on the best way to go about this - will I be able to scan/mux 192 leds? I also need to be able to controller each RGB LED via PWM - eep.

I've seen the Borg3D cube, but they used an FPGA rather than a uC.

Cheers

[view all 58 comments](#)