DMX Explained; DMX512 and RS-485 Protocol Detail for Lighting Applications

shabaz *24 Aug 2017*

Introduction

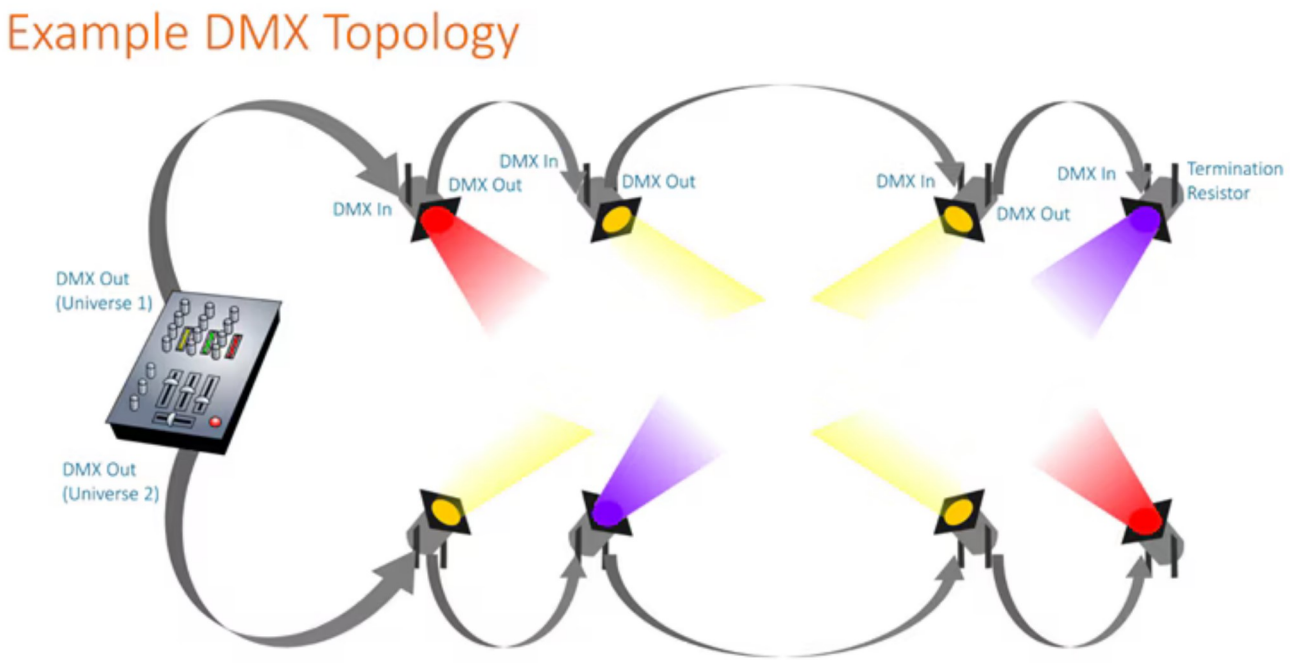
The Digital Multiplex (DMX) protocol (not the rapper :), also known as DMX512 or DMX512-A, is an industry-standard method of achieving lighting control, both manually (using a control panel) and for lighting automation (using a PC). It finds uses in nightclubs, restaurants and theatres.

Lighting controllers generate serial signals in a format called DMX512 (I'll use the term DMX for ease of reading throughout this blog post), and the lighting controller output is connected to a light fitting. Each DMX-capable light fitting has a DMX input, and a DMX output. It is a master-slave system where the lighting controller acts as the master, and the light fitting acts as a slave to the controller but as a master to the next light fitting that is attached.

The photo here shows an example; this is a single multi-color light fitting, and the 'DMX In' would be attached to a lighting controller or a PC. The 'DMX Out' is not attached in this example. It would connect to another light fitting.

The light fittings are therefore chained, and the lighting controller will be able to address each one. There is a limit to the chain length (512, but the limit is often lower for reasons explained later) and so many lighting controller will feature several (or more) DMX outputs, so that additional chains can be created. Each chain is known as a DMX Universe.

The diagram here shows what a lighting set-up with DMX can look like; this example has two DMX universes.



Having recently explored a USB adapter circuit to convert to serial (also known as a USB to 'universal asynchronous receiver/transmitter' bridge) intended to connect computers via USB to serial interfaces, I was curious to examine interesting serial protocols so we can create our own open source hardware and software. So, I explored the standard music interface, MIDI, with the CP2102N to have my music keyboard to act as a synth under control from the computer. Next I wanted to examine lighting. DMX is hardly an obscure protocol but I struggled to find good examples online of it. It is one thing to look at the protocol documentation, but another thing to examine implementations. Manufacturers sometime do things subtly differently, rendering some things inoperable with each other, or performing sub-optimally.

beacon_dave worked at Looking at the DMX 512 protocol with a Keysight EDUX1002A scope for a detailed analysis with real-world oscilloscope traces. He discovered some interesting things like unusually long delays in parts of the protocol. I too was keen to examine real-world equipment; I bought some Dialight LED modules ages ago, but I didn't know how to use DMX until I examined Dave's article.

Since I had a USB-to-UART bridge, I figured I could build a custom controller for the LED modules. But first I wanted to snoop the traffic and examine if it was similar to Dave's findings, or different in any way.

This blog post doesn't cover the design of a controller, but is intended to document the DMX protocol with real-world traces, to complement the work Dave did. Together there are now two practical examples of DMX implementations, so that it is possible to build solutions with high levels of interoperability.

What is DMX?

As mentioned earlier, it is a system to control lighting although other devices can be controlled too. The clue to how it works is actually in the name, 'Digital Multiplex'. The protocol mainly consists of a burst of data (known as a packet) where, much like a timeslot, the position of data in the packet defines what device the data is destined for. In other words, there is no address/data pair. Instead, the address is implied depending on the position in the packet.

The protocol is actually very simple. The bits and bytes are carried using a serial interface running at 250,000 bits per second, and the electrical interface is a balanced line pair (and 0V reference) often interfaced using 5-pin XLR connectors. The electrical interface is RS-485.

The ‘512’ in the DMX512 name is very descriptive too. It means that up to 512 usable data bytes can be sent in a packet (513 are sent, but the first is not used). A packet represents the entire information needed for a DMX universe.

If each lighting fixture just features basic single-color (e.g. white light) dimming capability then a single data byte can control the lighting fixture and allow up to 255 brightness levels from off (zero) to fully on (255), and so up to 512 devices can be controlled.

Some lighting fixtures have red/green/blue lamps inside, and they may need three data bytes for full RGB control. Since a packet (and hence the DMX universe) can only have 512 usable data bytes, this means that up to 170 RGB devices can be controlled, not 512.

What is RS-485?

RS-485 is a method for connecting up different devices. It uses three wires; two carry signals in a balanced format, and the third is a 0V reference wire. Multiple devices can be connected to this wiring bus although in practice with DMX lighting only one device is connected at each end; a master and a slave.

Since lighting could involve many devices, it is possible to have two separate RS-485 connections per lighting device. One connection acts as a RS-485 slave from the controller, and the other connection acts as a RS-485 master for connecting to another lighting device in the daisy-chain.

For now just a single slave device will be considered, and it will be controlled by a single master device (which would typically be a lighting controller device).

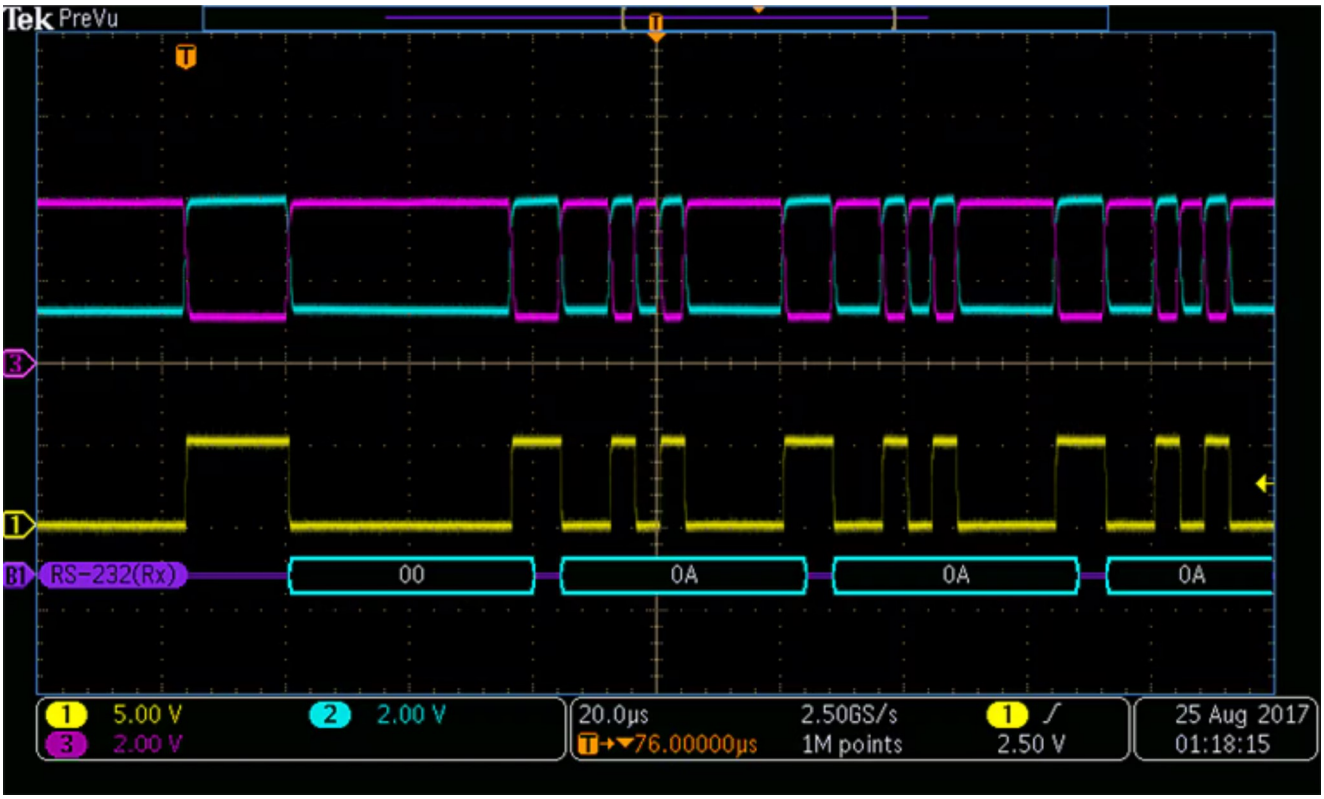
Back to the three wires; the two balanced signal wires are called **A** and **B** (or **Non-inverting** and **Inverting** respectively, or just marked ‘**+**’ and ‘**-**‘ (but be careful not to mix up with the DC power supply connections in that case!)) and have out-of-phase signals. In other words, when one of the signal wires is at a high voltage with respect to the 0V reference wire, then the other signal wire will be at a low voltage and vice-versa.

Since microcontrollers and other logic devices usually just have single-ended logic levels referenced to 0V, a device known as a RS-485 Transceiver is used to interface between the microcontroller (3.3V or 5V logic) and the RS-485 world which has the three-wire method outlined. Example RS-485 transceivers are MAX485 and ADM485; these are low-cost non-isolated transceivers but for a robust design suitable for use across buildings and large

premises, fully isolated RS-485 transceivers must be used (and it is mandatory according to the DMX specification for the input side to be fully isolated but be careful, don't assume all products will have isolation!).

A picture says a thousand words. The oscilloscope trace below shows in yellow the logic level signals corresponding to the data stream of values (in hexadecimal) **0x00**, **0x0A**, **0x0A**, **0x0A**. The bits/bytes will be examined later but for now we can examine the voltage levels. The yellow trace is a 5V logic level signal suitable for 5V microcontrollers.

The blue and purple traces are the A and B lines respectively. The blue (A line) trace is non-inverting, and follows the yellow trace (although the voltage levels are different). The purple trace has an opposite phase. The point of this is that there is a greatly increased threshold to protect against noise when the signal is received; this is the key advantage of balanced signals. The oscilloscope trace was taken by snooping the RS-485 communication from a commercial lighting controller. A MAX485 device was used as the receiver to perform the snooping operation and the blue and purple traces correspond to the signals taken from the pins marked A and B in the MAX485 data sheet, with reference to the 0V signal.



So, this explains the voltage levels. From here onward, only the yellow trace will be shown, and the line A and line B signals will not be shown. We can now examine the data format and what the bits/bytes mean.

Serial Communication

Now that the voltage levels have been discussed, we can examine how content is sent over the RS-485 system. The serial communications method entails sending bytes of data in a stream of eight bits per byte, wrapped around a **start bit** and **stop bits**. The start bit is a single bit period set to logic zero. The stop bits when using DMX are two bits set to logic 1.

The start bit, and the eight bits in the byte, and the two stop bits, are a total of 11 bits. It is known as a **frame**. The bits in the frame are sent 250,000 times a second (known as 250,000 baud) and this means that each bit is 4usec long.

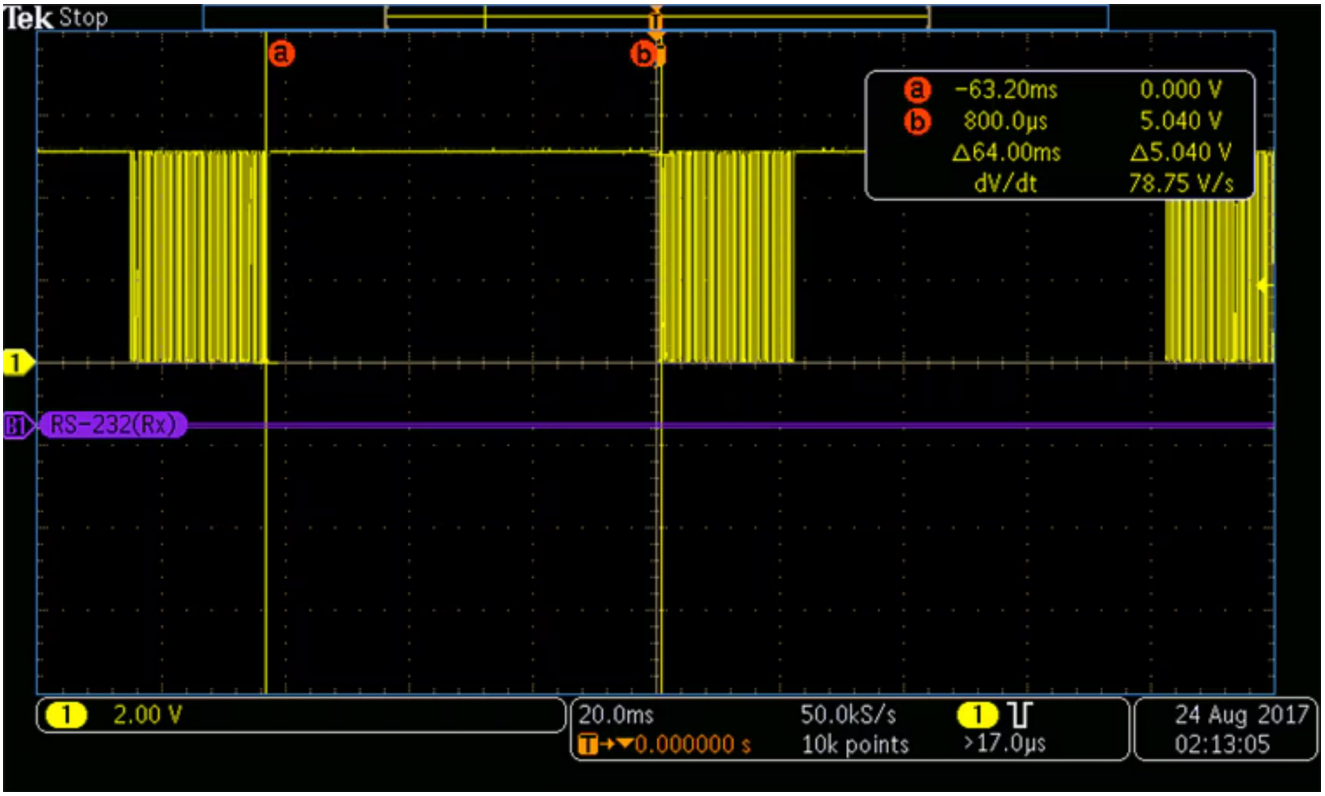
The DMX protocol will send a specific **start-of-packet procedure** on the serial bus, and then send a sequence of frames. Usually it will perform a start-of-packet procedure, send 513 frames, and then pause (idle) for a while and then repeat. It is explained in more detail next. By the way not all DMX controllers may send 513 frames, some may send less.

DMX Protocol Detail

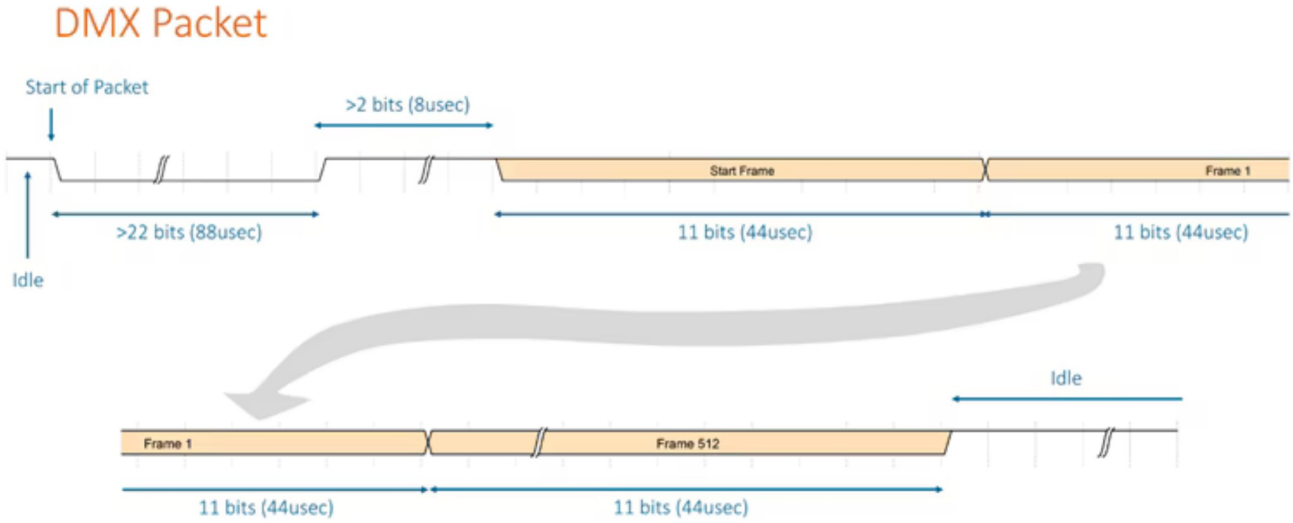
The default idle state of the bus is at logic 1 (a high voltage on the line A, and a low voltage on line B), and logic high (such as 5V or 3.3V) at the microcontroller side of the RS-485 transceiver).

With the equipment I tested, I saw that approximately every 70 to 90 milliseconds, a packet is sent (i.e. a start-of-packet procedure, followed by 513 frames) and then the line goes to idle state again. The 70-90 millisecond period can vary and the exact value is unimportant generally, except it will matter when lighting effects (like fading transitions) are required.

The screenshot here shows three packets and the long idle periods in-between.



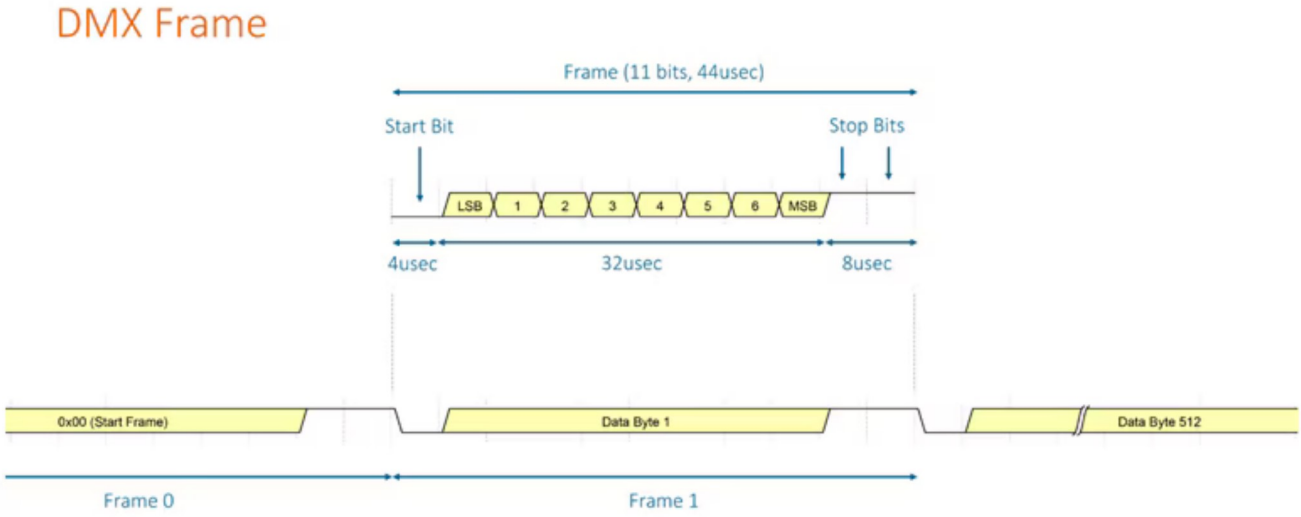
The diagram below (click it to enlarge for detail) shows one entire DMX packet, starting with the line idle (logic 1) and ending with the line idle again. This will repeat every 70-90 milliseconds with either the same packet content if the lighting levels are to remain the same, or with changed packet content if lighting effects/transitions are occurring. It follows that just over ten discrete steps per second are therefore possible when performing lighting transitions, if the packets are being sent every 90 milliseconds. If no packet is sent, or if there is a long delay between packets, then the lights will remain at the previous set level until the levels are updated by new packet content.



The start-of-packet procedure is a logic zero for more than 22 bit periods,followed by a logic 1 for more than 2 bit periods.

The packet represents 512 different light levels. In other words, per packet, 512 devices could be addressed. There are 513 frames, and the first frame (frame zero) doesn't correspond to a real device. For lighting applications it always contains a hard-coded data byte set to zero, and is known as the start code or the start frame. Apart from the fact that the data byte inside this first frame is set to zero, it looks just like any other frame. Note that if this start frame's byte contains any value other than zero, then this means that the packet is being used for a non-light application such as pan/tilt. The entire packet should be ignored in that case.

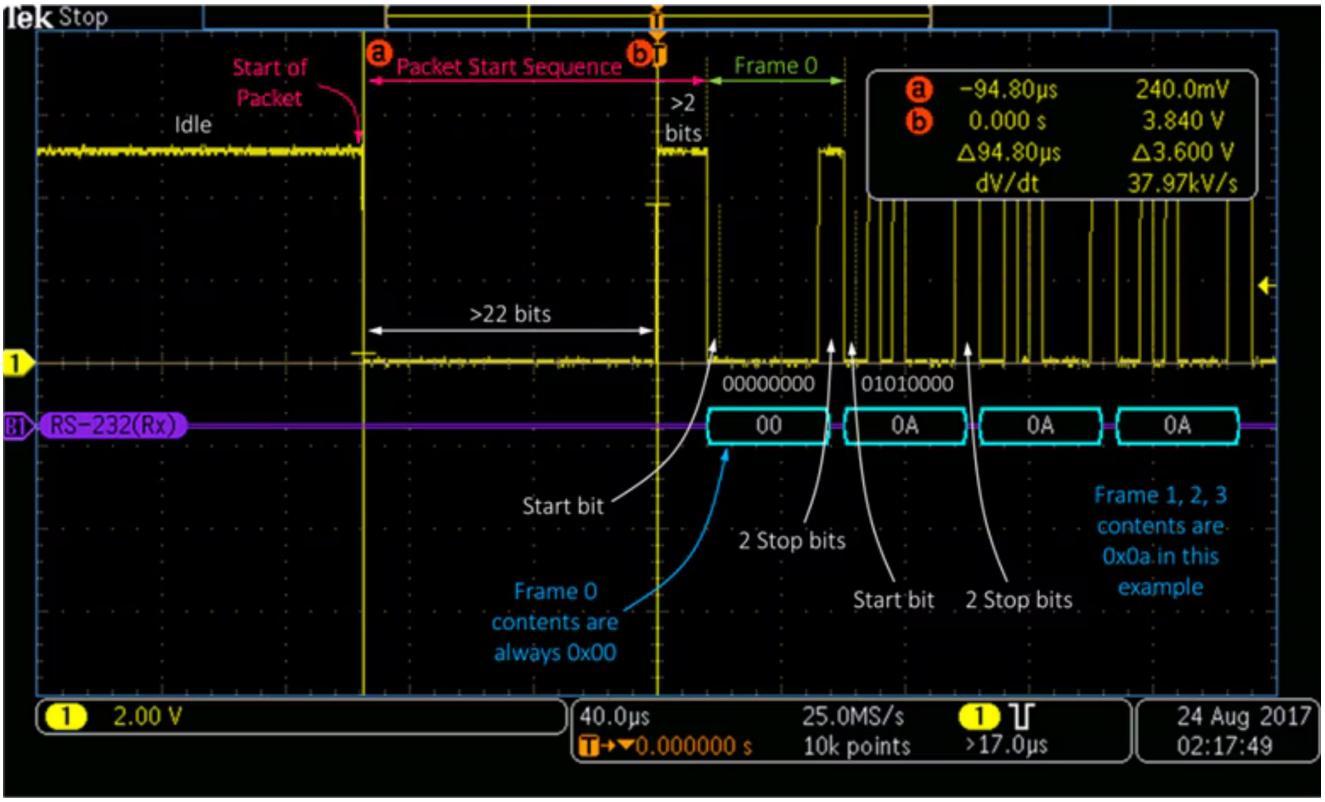
If we look inside any frame, we can see that it consists of just a single byte wrapped around one start bit and two stop bits. The start bit is zero, and the two stop bits are high.



The packet's first frame byte content is always 0x00 for light applications, and then a further 512 frames are used to select the levels of 512 lights. These are logical lights, since each physical light fitting may contain red-green-blue LEDs, or even red-green-blue-white. That being the case, the 512 logical lights can be used for (say) 128 physical lights if each one is consuming RGBW values. If there are just RGB lights (i.e. three channels or frames used per physical light fitting) then up to 170 physical lights could be supported. If there are just single-color white lights then up to 512 of them can be supported.

The packet starts with the start-of-packet procedure which as mentioned is the bus going to logic level 0 for more than 22 bit periods (88usec; in practice I saw 94usec) and then high for more than two bit periods (8usec but in practice I saw around 16-24usec). After that, a total of 513 frames are transmitted, and each frame contains a logic zero start bit, eight data bits and two logic '1' stop bits.

The annotated 'scope trace below shows the >22 bit-period logic low level, followed by the >2 bit-period high level, followed by the frames containing 0x00, 0x0A, 0x0A, 0x0A. These are the start frame, and frames 1, 2, 3.

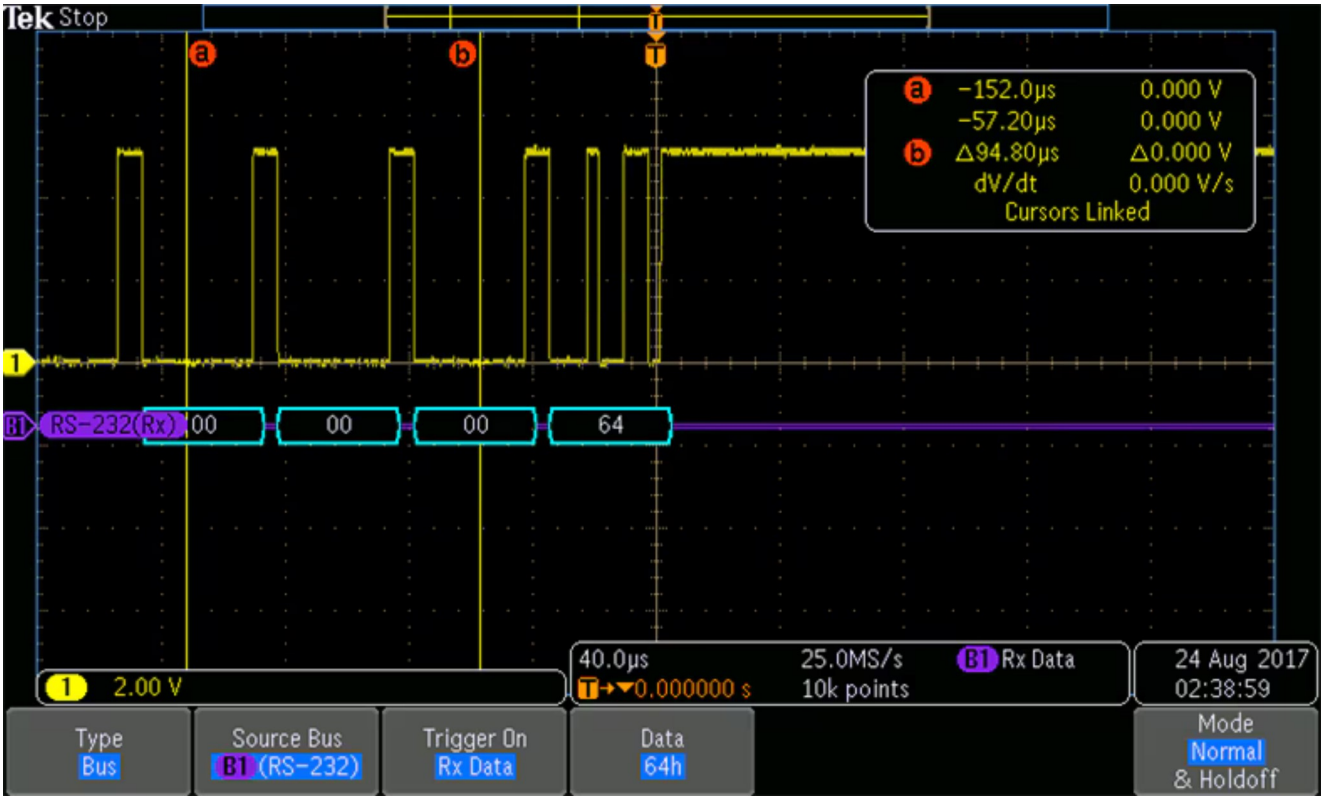


The first frame has a byte that as mentioned is 0x00 for lighting applications (ignore the entire packet if you see a different value!), and then the subsequent 512 frames contain the light levels. All bytes are sent least significant bit (LSB) first. The trace screenshot above shows that the first three light levels are set to 0x0a (decimal ten) which is quite dim (the range is 0 to 255).

Incidentally the frame number is known as the address in DMX terminology. So, in the oscilloscope trace above, DMX addresses 1, 2 and 3 are set to 0x0a (decimal 10). They are also referred to as ‘channels’. It is all a bit vague :) but the terms represent the same thing.

Since each frame consists of a logic zero start bit, then eight data bits and then two logic ‘1’ stop bits, this means that each frame takes up a total of 44 usec (4 usec to send the start bit, 32usec to send the 8 data bits, followed by 8usec total for two ‘stop’ bits). Immediately after this the next frame is transmitted in the same manner.

After all 513 frames have been sent, the bus idles high for about 50 to 70milliseconds and then another packet is sent. The ‘scope trace below shows the last few frames at the end of the packet. The last frame content happens to contain 0x64 (decimal 100) in this example.



As mentioned, if the same frame per packet contains the same information then the lights remain at a steady level. If the packets stop, then the lights still remain at a steady level, therefore the time between packets doesn't matter for a steady setting. For effects (e.g. lighting transitions) then each packet would incrementally adjust the lighting levels.

One other point worth mentioning is that brightness levels from 0 to 255 are not very granular. It is possible to see jumps in transitions, they won't be smooth. For that reason, some lighting fixtures will use two bytes to set the level. The first byte in the pair will be the crude brightness level (the 8 most significant bits of a 16-bit word) and then the second byte will represent the fine brightness level (the 8 least significant bits of the 16-bit word).

Another use for the DMX bytes could be to adjust pan/tilt settings of motorized light fittings.

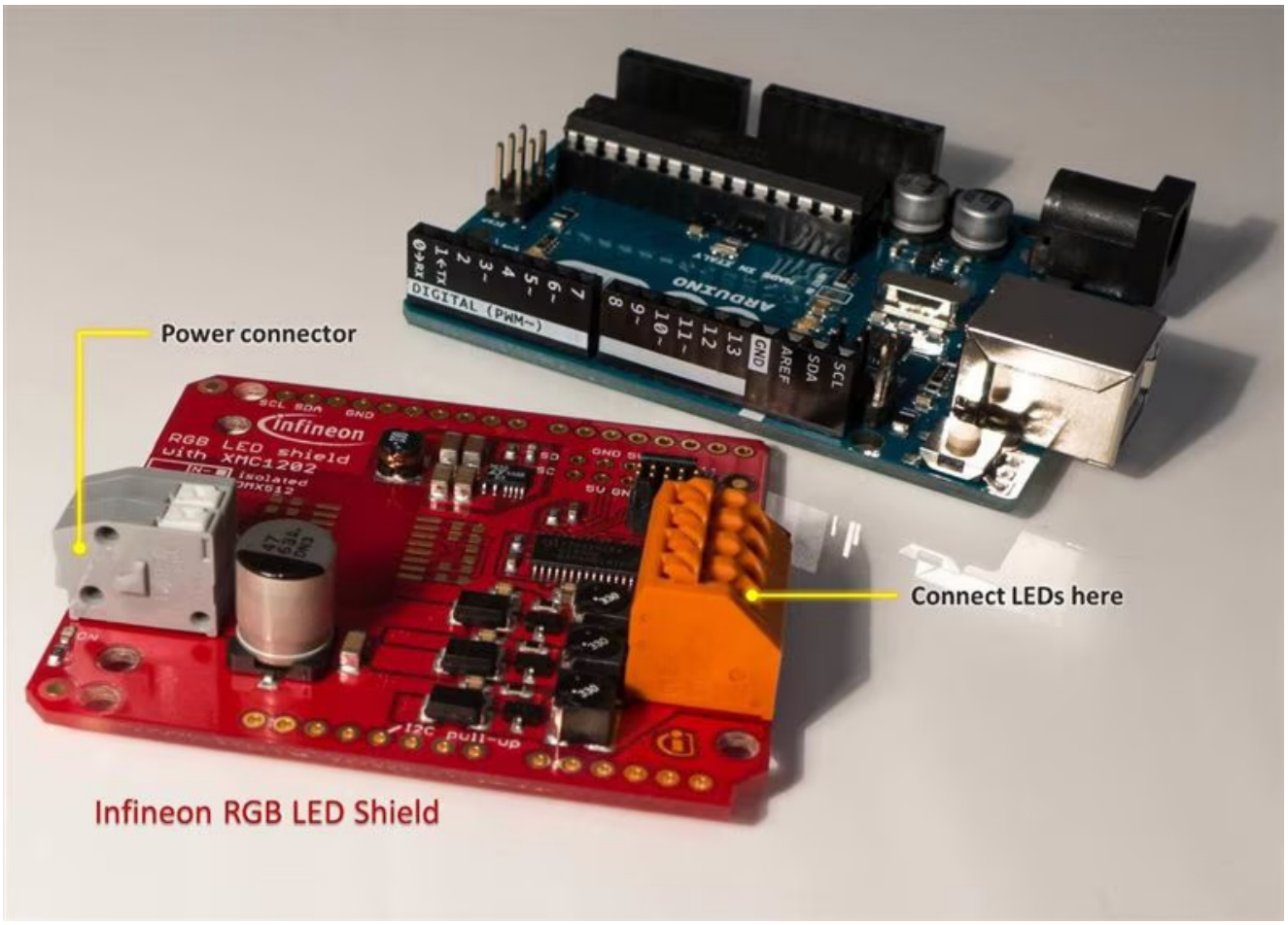
Summary

DMX512 is an easy-to-use protocol. Hopefully with the description in this blog post, and Dave's article, it should be more straightforward now to develop lighting automation projects, or to find new uses for the DMX protocol for controlling different devices.

For low-cost experimentation, the Infineon RGB LED Shield for Arduino is very useful; it has terminals to connect up RGB LEDs and space on-board to solder an RS-485 transceiver chip. The on-board Infineon microcontroller can be programmed to interpret the DMX

packets and control the LEDs.

Incidentally it could also be used as a lighting controller, if no LEDs are attached to it, provided the software is changed to operate in this manner as a DMX Out (RS-485 Master) device, perhaps under the control of the Arduino board that it plugs onto.
See here for more information on it: RGB LED Shield from Infineon - Getting Started Guide .



beacon_dave *over 7 years ago*

shabaz - This may be of interest, Microchip appear to have added DMX-512, DALI, & LIN protocol support to the UART in their newer PIC 8-bit microcontrollers.

DMX-512 using the UART with Protocol Support

http://ww1.microchip.com/downloads/en/AppNotes/TB3204-DMX-512-using-UART-90003204A.pdf

DMX Controller using the UART with protocol support module

https://mplabxpress.microchip.com/mplabcloud/example/details/517

DMX512 Receiver using PIC18FxxK42 with DMA

https://mplabxpress.microchip.com/mplabcloud/example/details/382

It looks like you can also use the Direct Memory Access module with this new UART to off-load even more tasks from the core processing.

Unfortunately Microchip don't appear to have added the same UART functionality to the likes of the AVR ATMEGA4809 as used in the new Arduino UNO WiFi.

For those interested in WS2812 pixel LEDs, then the PICs Configurable Logic Cell module may also be of interest.

Using the Configurable Logic Cell (CLC) to Interface a PIC16F1509 and WS2811 LED Driver

http://ww1.microchip.com/downloads/en/AppNotes/00001606A.pdf

8x32_NeoPixel_LED_PIC18FxxK42

https://mplabxpress.microchip.com/mplabcloud/example/details/378

shabaz *over 7 years ago in reply to beacon_dave*

Hi Dave,

> Thanks for the info! That's a shame they didn't implement it for the ATmega from my biased perspective because I'm more familiar with AVR than PIC for 8-bit devices. In the ARM world, the Infineon chips have DMX support, so this is another option, although I'm guessing the PICs will be very low-cost for high-volume products.

neilk *over 5 years ago*

Hi shabaz

I've only just read the original blog post and found it fascinating!

As a student in the late 60s, I got involved in theatre lighting, from ancient rheostat systems through to quite sophisticated thyristor based boards. I gave up my involvement with amateur

dramatics in the mid 70s. Since then I've watched the increasing level of sophistication in lighting control systems and the introduction of LEDs and remote control pan and tilt.

I've often wondered about the underlying control protocols - now I know!

Thanks again for posting this

Neil

shabaz *over 5 years ago in reply to neilk*

Hi Neil,

Thanks!

Probably many older theaters still use those massive 'dimmer packs' or whatever they were known as, and the 0-10V? control. I recall the local theatre near me, despite it being tiny, having a toasty room with huge stack of those things to control the incandescent lights/halogens or whatever they used :)

neilk *over 5 years ago in reply to shabaz*

Hi shabaz

Yes! The dimmer packs got pretty warm, but not as warm as the incandescent or halogen powered lanterns!!

The last system I worked on was indeed a load of dimmer packs in the wings and a control desk at the back of the auditorium, the two being connected by a big fat multicore cable. So great to have the audience view of the lighting effects!!

Neil

beacon_dave *over 8 years ago*

Nice write-up Shabaz.

Just a quick heads-up but the packet's first frame byte (AKA 'the start code'), is not always 0x00.

There is a list of known alternate start codes in use available on the Entertainment Services and Technology Association (ESTA) web site at:

http://tsp.esta.org/tsp/working_groups/CP/DMXAlternateCodes.php

People may want to ignore packets with start codes that are not 0x00 as it can send erratic movements to PTZ motors in moving fixtures.

shabaz *over 8 years ago in reply to beacon_dave*

Hi Dave,

Thanks! Thank-you for examining it. As you say, other values for that first frame are possible but then the packet should be ignored. I'll add text to specify that into the body of the article right now.

Thanks for the link too, this is awesome to see what other codes there are, it is really useful information.

beacon_dave *over 8 years ago in reply to shabaz*

There is some more reading material available at:

TSP - Published Documents - About TSP Documents, Published Documents, Public Review Documents, Procedural Documents

jc2048 *over 8 years ago*

Nicely done - some useful information there. I found this section a bit ambiguous.

> *RS-485 is a method for connecting up different devices. It uses three wires; two carry signals in a balanced format, and the third is a 0V reference wire. Multiple devices can be connected to this wiring bus although in practice with DMX lighting only one device is connected at each end; a master and a slave.*

> *Since lighting could involve many devices, it is possible to have two separate RS-485 connections per lighting device. One connection acts as a RS-485 slave from the controller, and the other connection acts as a RS-485 master for connecting to another lighting device in the daisy-chain.*

I know what you meant, but a diagram would have helped to emphasise that the bus is multidrop, with the lamps 'T'-ed off of it (even though the 'T' is normally hidden inside between the 'in' and the 'out' connectors). As it stands, someone might intepret what you written as the fixture buffering the signal and then sending it on. It needs to be multidrop to allow for RDM (Remote Device Management) where the fixture can transmit data back to the desk/controller. You might also mention that strictly, to conform with the spec, the input should be isolated (to avoid problems with ground loops in a large scale installation) - obviously, for a personal project, you can just ignore that, but for a commercial product you'd need to work to the spec.

But the main point you're making, that DMX512 is a simple protocol that's easy to generate and decode, is spot on and hopefully you'll encourage some activity here in that area. (It's easy to generate because you're just throwing bytes at a UART and then slapping in a long break every once in a while. It's easy to decode because you just have to recognise the break, check the start code, and then count off received bytes until you get to the ones you want.)

If 'something in a tin box' wins for September, perhaps someone could have a go at 'DMX512 in a Baked Bean Can'.

shabaz *over 8 years ago in reply to jc2048*

Hi Jon,

Awesome idea for DMX-in-an-Altoids-can :)

And really good points. I'm hoping to tear down some devices to examine if some utilize RS-485 multidrop, because the one I have does not, so I can't see if this is typical or a typical currently : (

The one lighting fixture I grabbed information from (an end-of-sale Dialight product, that I had to reverse-engineer due to lack of information) is not multidrop, it has two separate RS-485 transceivers, i.e. it must act as a slave on one connector, and as a master on another connector.

I believe the internal behaviour of the device is to implement the forwarding in software, so it is a daisy-chain type arrangement which allows for some flexibility in how devices in the chain are handled depending on their configuration.

The devices furthest away are still controlled by the controller, but the information makes its way via multiple RS-485 masters and slaves. This would have some advantages, like devices could selectively decide to forward information, or to modify content, e.g. move the data bytes in the packet (much like moving the TDM byte positions) such as a shift-left operation, so that devices could be auto-addressed even if they are all configured identically. It could allow the possibility to ship all devices to a site with the same configuration, and the installer does not need to bother with configuring each one separately.

I'm still not sure if this is typical, because I have only examined one device. It could be that this device implements the separated buses but other vendor ones might save on a transceiver chip and some reduced software functionality, but I have limited exposure since I've just got the Dialight one to examine.

We could be in the realm of what the spec says should be supported, but what the devices support can include either proprietary stuff, or practical things like separated bus segments i.e. separate RS-485 masters and slaves.

I hoped to capture the real-world observation so that we have some devices to compare to. You're right on the isolation, I'll clarify the mandatory-ness in the blog post, but interestingly the device I examined did not implement isolation. I suspect the reason for that is the use-case and/or age of the product, perhaps it was likely intended for integration into a larger product (e.g. a product where multiple DMX-addressed devices are already wired together internally), where the isolation could occur at a single transceiver, i.e. the external connector to the product.

beacon_dave *over 8 years ago in reply to shabaz*

I think you will find that the majority of the lower end kit uses a multidrop bus with the in out connectors just Tee'd together.

Some of the higher end stuff I believe is hybrid - there is a relay inside the fixture which breaks the Tee and routes the signal through processing. If the fixture fails and loses power then the rely reverts the signal path back to a Tee so as other devices on the bus can still function.

Probably worth mentioning that some fixtures are auto terminating whereas others require a terminator resistor on the last device on the bus.

Probably also worth mentioning that the spec requires the use of 5 pin XLR connectors where as many commercial fixtures use 3 pin. The original spec had two differential buses for full duplex operation, however it turned out that RDM type features never used it. The 3 pin probably became popular as it allowed people to use the same cables for microphones and lighting. Although a lot of microphone cables aren't rated for DMX use and is notorious for causing problems in lighting set-ups.

beacon_dave *over 8 years ago in reply to shabaz*

The DMX-in-an-[altoids|baked bean]-can sounds like a great idea but it may need a little bit of a 'helping hand' from the community.

A DMX project effectively has two parts - the controller and the device being controlled. If you are designing a controller project then you need a device to be controlled to test it against and if you are building a device to be controlled project then you need a controller to test it against.

Not everyone will have access to a pile of DMX equipment to test against, so in order to get this idea moving, it may need a couple of low cost reference designs that people can build first to use to test their project design against.

jc2048 *over 8 years ago in reply to shabaz*

Sorry, I'm remembering it wrongly on the isolation. It's not mandatory, it's just "preferred".

> *4.6 Preferred method of earth grounding data link common*
> *DMX systems should make use of earth ground referenced transmitting devices and isolated receiving devices.*

The specification goes into considerable detail on the variations for grounding on transmitters and receivers in an annex.

jc2048 *over 8 years ago in reply to beacon_dave*

Well, that's you sorted for your piece of DIY Test Equipment - do a DMX512 test generator. You could knock that up with an Arduino in no time.

beacon_dave *over 8 years ago in reply to jc2048*

I have been pondering over something along those lines for a little while. Was thinking of a test device in 'bare-bones' configuration that outputs:

- DMX channels 1 to 255 outputting values 0x01-0xFF
- DMX channel 256 value 0x00
- DMX channels 257 - 512 outputting fade up/fade down at different rates

Then by adding a selector switch, it adds to the above by varying the number of channels being output:

- single channel with packet padded out to DMX standard minimum packet length of 1,196µs
- single channel max refresh test packet length of 184µs
- 24 channels, DMX standard minimum packet length of 1,196µs

- 32 channels, 64 channels, 128 channels, 256 channels, 512 channels
- etc.

Then by adding some potentiometers to the analogue inputs it then gives user configurable values of 0x00-0xFF for the first few channels (e.g. up to 6 faders with an Arduino UNO)

And finally by adding a small LCD display with cursor pad buttons it allows the user to:

- over-ride the default timings for break, mark after break, mark between frame, mark after last channel
- custom set some channel values
- custom set fader min max range
- etc.

People can add as little or as much of the above functionality as they require.

May need to be careful if attempting it on the likes of the popular Arduino UNO platform due to the limited memory (if attempting to work with all 512 channels in a user-configurable type mode) and the limited number of USART ports if you wanted to use the Arduino serial console as a user interface and the USART for generating the data frames. However should not be an issue if dynamically generating most of the data frame values and 'bit-banging' the output pin.

▼ View More

genebren *over 8 years ago*

This was a really nice article. Back in July, I brought on a new client who wanted me to build him a new animatronics controller based on DMX-512A. I had never heard of DMX so I did a quick bootstrap on the subject and got started. I did not take too much time to understand the basics, but it took way longer to fully understand that not all products out there followed the specification. I found that I had to be very flexible in how I interpreted the timing, especially the start-of-packet timing.

I have since completed that project and have moved on to a couple of related products and enhancements to the first one.

It is always nice to learn something new and even better to be able to build on what you learned.

Gene

▼ View More