Ballast emulator

The projector did not turn on without a lamp, and I thought it would be wiser to emulate the ballast instead of adding a dummy load.

USHIO ballast

The ballast was an USHIO type, picture of which is below.

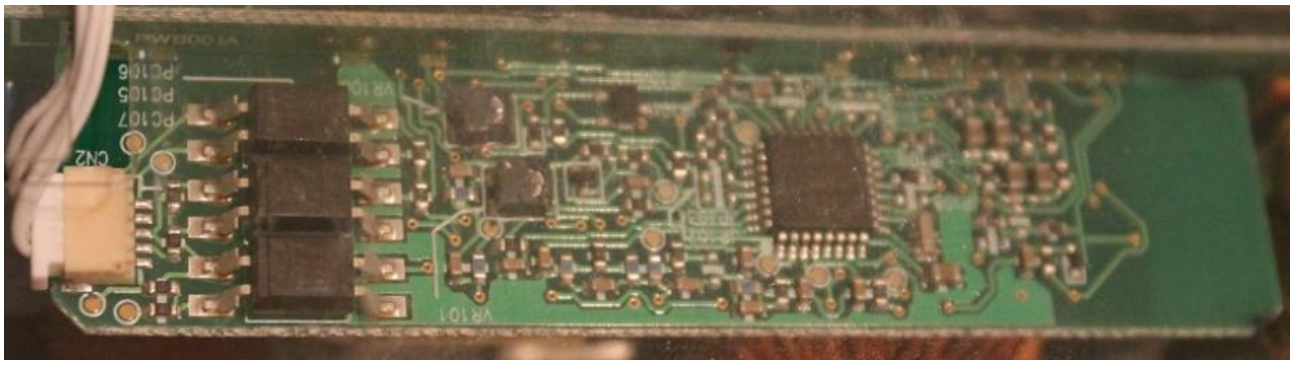

Figure 1: USHIO ballast communication board

On the left side there are 3 optoisolators; two isolate signals from projector to ballast and one is used for replies from ballast to projector. The basic schematic of the projector side of the optoisolators looks like Figure 2.

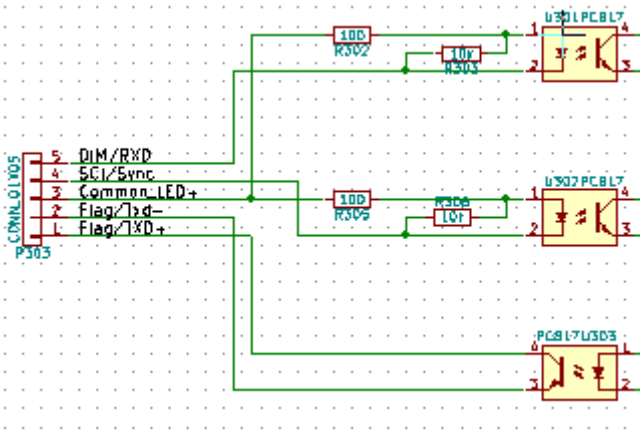

Figure 2: Ballast interface, projector side of optoisolators

I made a similar optocoupler setup on a breadboard, which had pin headers for all the signals. I used two 1.5 V batteries in series to power the secondary (safe) side of the optocouplers with about 3.2 V, and probed the receive signals with an oscilloscope. Looking at the connector pins (on the left of Figure 1), I go forward and name the ballast receive signals so that signal in wire 5 is DIM/RXD and signal in wire 4 is SCI/Sync.

Table 1: Pinout of OSRAM standardized ballasts, similar pinout in USHIO

control board connector CN2	**PIN 1 : Flag / TxD+** (Phototransistor Collector)
	PIN 2 : Flag / TxD- (Phototransistor Emitter)
	PIN 3 : Common LED+ (LED Anodes)
	PIN 4 : SCI / Sync. (cathode LED)
	PIN 5 : DIM / RxD (cathode LED)

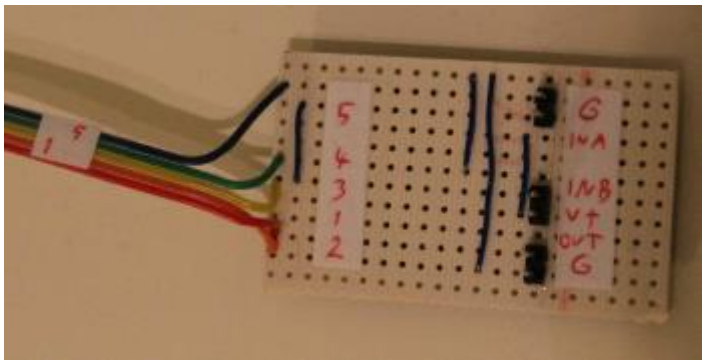

Figure 3: Optocoupler board, top side

When marking the connectors, I did not know the names so I named wire 5 (DIM/RXD) as IN A (input to microcontroller), wire 4 (SCI/Sync) as IN B and the last one I simply called “out”. G is ground and V+ is the supply voltage to the optocouplers, 3.3 V or 5 V works fine.

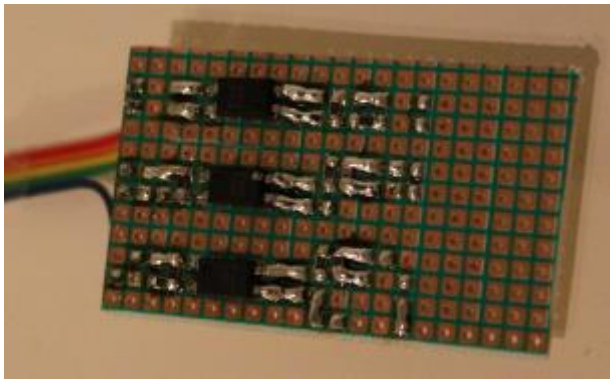

Figure 4: Optocoupler board, bottom side

Later I added an Attiny85 microcontroller to the empty space on the board.

Measurements

I measured all the signals from the microcontroller side of the optocouplers.

First, I measured that when the bus is idling, both receive signals are high. Then as soon as the projector powers up, the SCI/Sync signal goes low. This signal stays low as long as the projector is powered (I think), so I guess it is some kind of powerup request.

Then I started analysing the data on the DIM/RXD pin. There was some kind of serial communication, and since it is only one signal, I assumed it is UART. The first message after the projector boots, and long after the SCI/Sync goes low looks like this.

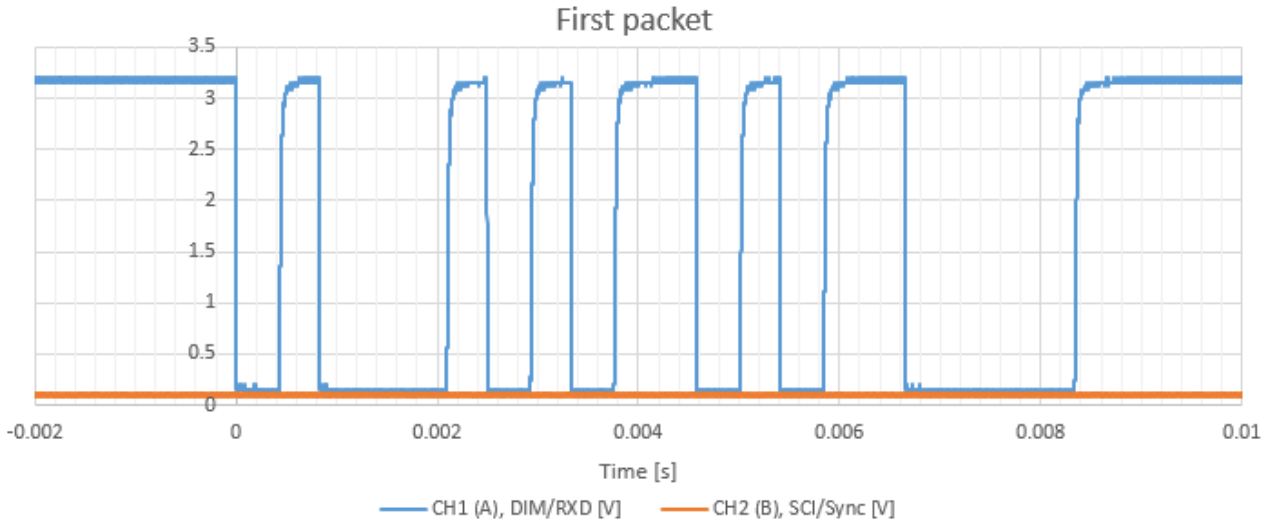

Figure 5: First message from projector

The projector keeps sending this message for maybe 5 times every 2 or so seconds, and then it starts blinking an error (lamp error I assume).

Looking at the data, first the bit length is 418 us, which indicates 2400 baud:

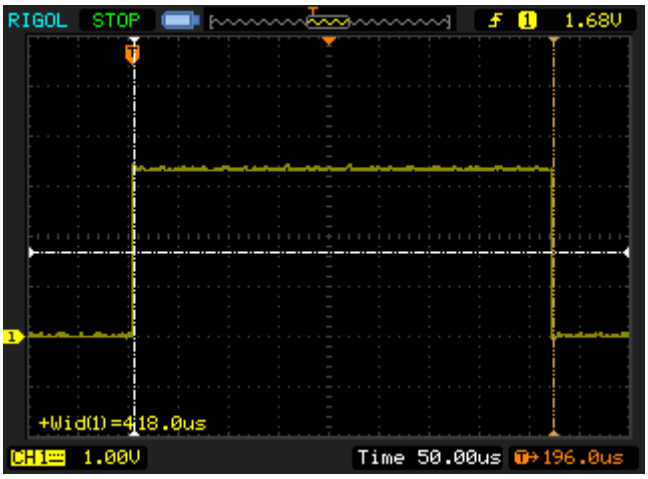

Figure 6: Single bit length

Then parsing the first message, the bits are 0100 0101 0110 1011 0000. Since I didn't know the protocol, I searched the internet for similar projects.

A user called Robertg at Eevblog forums [1] had measured a working projector and got the same serial message. He also had sniffed other messages and their replies. This made my project much easier, without measurements from a working projector it would have been hard to figure out what to reply to this message.

Communication

I copy the messages from Robertg's post at Eevblog forum [1] to here for further analysis. Below is his oscilloscope picture from the first message after the projector boots and the ballast's reply.

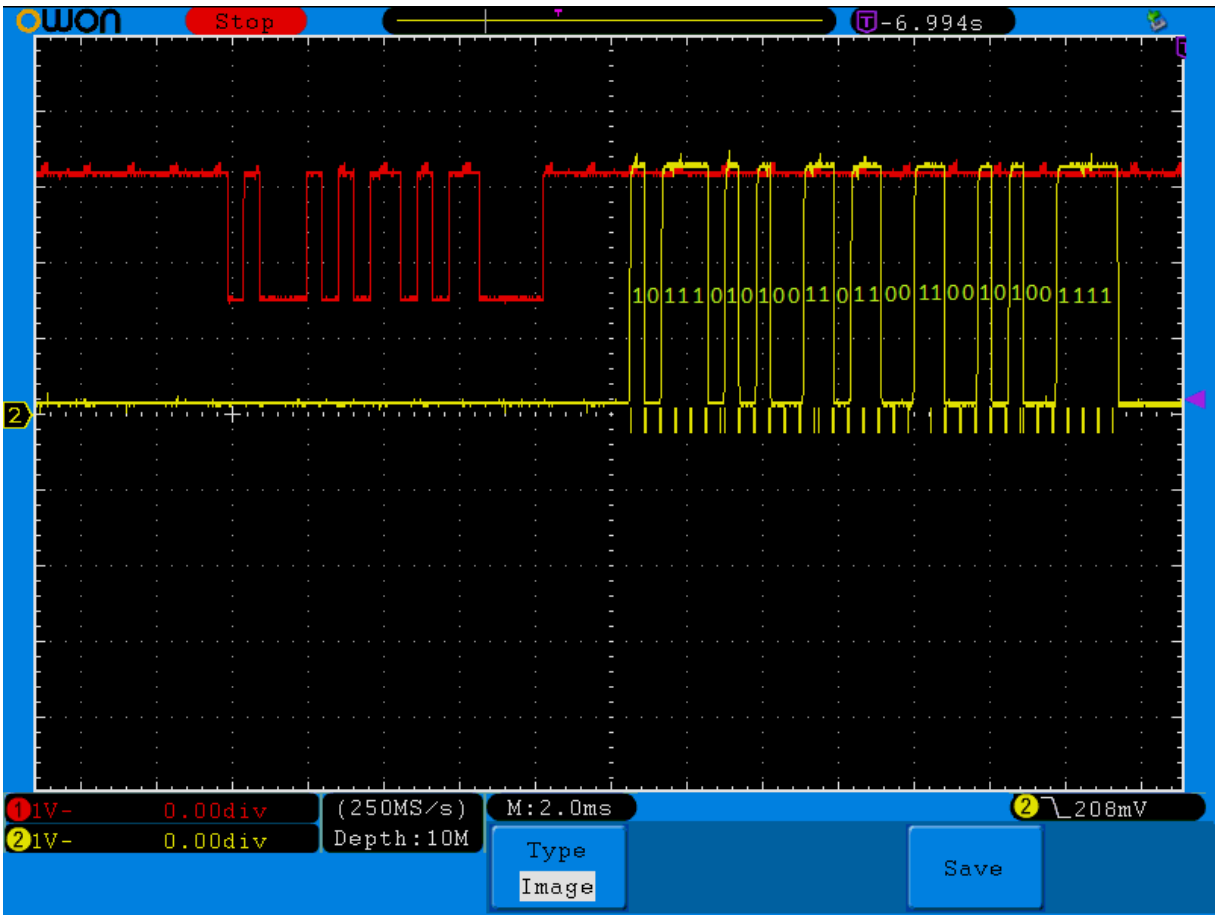

Figure 7: Robertg's picture of the first message and reply to it

Robertg figured out following bit patterns, assuming 1 is high and 0 is low, copied from the EEVBlog thread:

Table 2: Projector messages and replies, raw data

Projector sends	Bits	Ballast replies	Bits
01000101011010110000	20	10111010100110110011001010001111	31
00011001011001100010110101100000	31	10111110110101001111	20
00000101001010110000	20	11111010110110011101001010001111	31
01000101011010110000	20	10111010100110110011001010001111	31

We can see that messages are either 20 or 31 bits long, and projector messages end with 0000 and ballast replies with 1111. We can assume that 20 bit messages are 2 bytes and 31 bit messages are 3 byte.

UART typically has start bit and stop bit, so at least 2 extra bits per byte, and one byte is 10 bits. This does not make sense with the 31 bit messages though, they should be 30. I found some discussions on the internet [3] that the format would be 2400 baud, 9 bits, 1 stop bit. Then the messages should be 22 bits and 34 bits… It took me some time to figure this out by mostly trial and error.

Let's look at the projector side first. The bus idles high, so the first low going bit is the start bit. Then I tried different combinations of bits to figure out which makes the next byte start at low bit too. I found that after 9 bits, there is always 1 then 0, so I figured it is 1 stop bit high, then next start bit low. This matched the first

byte. Quite soon after it occurred to me that there is an even parity bit before the stop bit, and that makes the second or third byte look too short! I wrote the above bit strings according to this scheme. I also figured that the ballast side is inverted due to the logic in the optocouplers, so I inverted it in table below:

Table 3: Projector messages and replies, fixed

Projector sends												**Ballast replies**											
S	D0	8	PX	S	D0	8	PX	S	D0	8	PX	S	D0	8	PX	S	D0	8	PX	S	D0	8	PX
0	10001010	11	0	10110000	**11**							0	10001010	11	0	01001100	11	0	10110000	**11**			
0	00110010	11	0	01100010	11	0	10110000	**11**				0	10000010	01	0	10110000	**11**						
0	00001010	01	0	10110000	**11**							0	00001010	01	0	01100010	11	0	10110000	**11**			
0	10001010	11	0	10110000	**11**							0	10001010	11	0	01001100	11	0	10110000	**11**			

I denoted start bit with S, stop bit with X and parity bit with P. Also, bold bits are ones that are not clearly visible on the oscilloscope figures, or in the first data table, since they are at the same level as where the bus idles.

Now both messages are the same format, levels correct and parity matches. It's not clear which bit is most significant or should the bits be inverted, but I decided that bytes are sent with least significant bit first. From this data we can figure out following messages and replies:

Table 4: Ballast messages

Query from projector	**Reply from ballast**	**Comment**
51 0D	51 32 0D	
4C 46 0D	41 0D	
50 0D	50 46 0D	
4C 45 0D	41 0D	This is probably wrong reply

[1] http://www.eevblog.com/forum/beginners/video-projector-ballast-bypass-help/

[2] http://4hv.org/e107_plugins/forum/forum_viewtopic.php?109716.0

[3] https://forum.arduino.cc/index.php?topic=409298.0

[4] http://www.kerrywong.com/2016/06/05/hacking-an-osram-p-vip-projector-lamp-driver-board/

Debugging with TI TIVA launchpad

I connected the optoisolator board to a TI TIVA Launchpad [5] and coded a simple ballast emulator with that. This was quite similar to how I did the final version with Attiny85.

Following figure shows the debug session with the launchpad connected to the optoisolator board, which is soldered to the projector. I also have oscilloscope probes connected to the serial lines so I could verify the communication.

The Tiva software also echoed all the messages through the USB debug interface so I could monitor the messages with a PC. With this setup I noticed that some of the messages in the table above were most likely not correct, since the projector started spamming new messages rapidly.

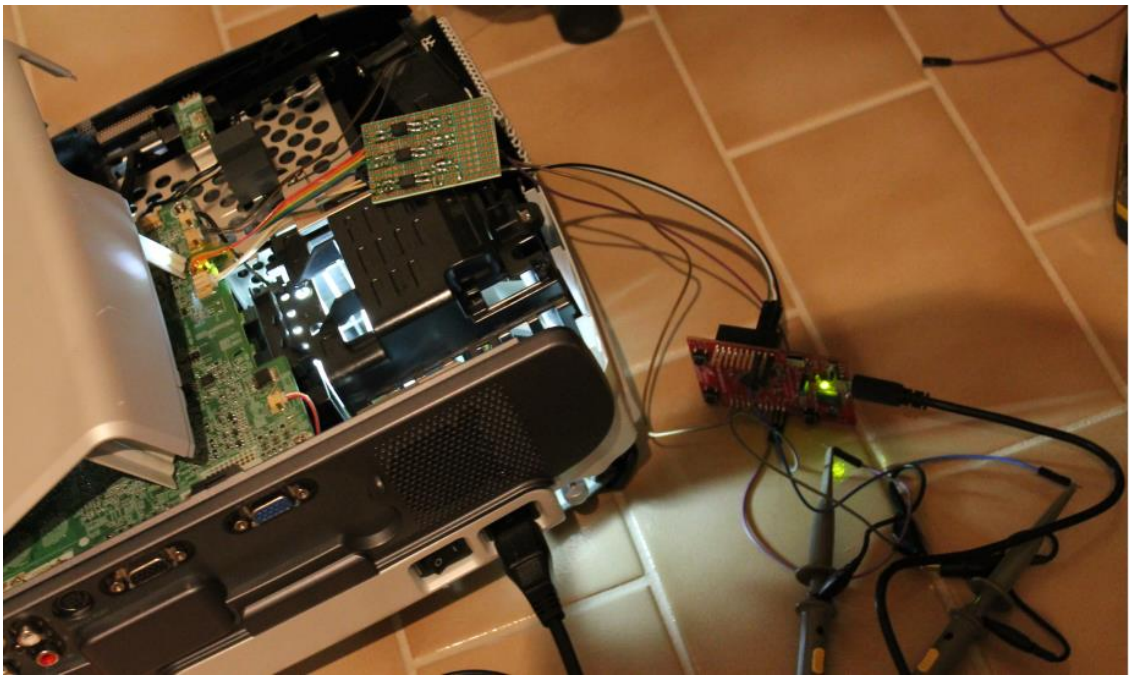

Figure 8: Prototype ballast emulator with Tiva launchpad

[5] http://www.ti.com/tool/EK-TM4C123GXL

OSRAM Protocol

This is not implemented yet.

OSRAM has published their ballast serial protocol description online [6], so I though it would be neat to also support this protocol. But since I have no way of testing the system, I decided to not implement this at this time. It would be quite straightforward to do if I could test the code.

[6] https://www.ercankoclar.com/wp-content/uploads/2018/04/standardized-uart-protocol.pdf

3-wire (flag) protocol

This is the simplest protocol. There is no communication, instead the pin states indicate whether lamp should be on or dimmed and response is if lamp is on. Following table shows the message states.

Table 5: Flag mode commands and replies

Command	Function	Reply
SCI/Sync high	Turn lamp off	Flag high
SCI/Sync low	Turn lamp on	Flag low
DIM/RXD high	Lamp full power	None
DIM/RXD low	ECO mode	None

Basically, we just need to reply to the turn on command by flipping the flag pin.