MASTERs 2013

The premier technical training conference for embedded control engineers

17043 LCP

Lighting Communication Protocols

DMX512 and DALI

Agenda

- Lighting Protocols Overview
- DMX512
- Lab 1 - DMX512 Controller
- Lab 2 - DMX512 Receiver
- DALI
- Lab 3 - DALI Control Device
- Lab 4 - DALI Control Gear
- Large Lighting Network
- Lab 5 - Implementing a Larger Lighting Network
- Summary

Goal for the day

- Understand the basic principles of DMX512 and how to use our Library
- Understand the basic principles of DALI and how to use our library
- Gain a basic idea on what a large lighting network is and some issues

Agenda

- **Lighting Protocols Overview**
- DMX512
- Lab 1 - DMX512 Controller
- Lab 2 - DMX512 Receiver
- DALI
- Lab 3 - DALI Control Device
- Lab 4 - DALI Control Gear
- Large Lighting Network
- Lab 5 - Implementing a Larger Lighting Network
- Summary

Lighting Protocols Overview

- **History**
  - Oil Lamps/Candles
  - Gas Lamps
  - Electric Bulbs with switches
  - Resistor control of bulbs
  - Variable transformer control
  - SCR's and TRIACS created
  - Analog Control
  - Digital Control (PWM / Serial)

Lighting Protocols Overview

- **Uses**
  - Domestic
  - Commercial
  - Theatrical
  - Architectural
  - Metropolitan

Lighting Protocols Overview

- **We will concentrate on the following**
  - **DMX512**
    - Theatrical & Architectural
    - World wide
  - **DALI**
    - Commercial (Office, Hotel, Conference)
    - World wide
    - Most popular in Europe & Asia Pacific
    - Growing in popularity in the USA

Agenda

- Lighting Protocols Overview
- **DMX512**
- Lab 1 - DMX512 Controller
- Lab 2 – DMX512 Receiver
- DALI
- Lab 3 – DALI Control Device
- Lab 4 – DALI Control Gear
- Large Lighting Network
- Lab 5 – Implementing a Larger Lighting Network
- Summary

DMX512
Introduction

- **DMX512 – "Digital Multiplex with 512 pieces of information"**
- **Designed for theatrical lighting**
- **Needed a more reliable protocol than 0-10V**
- **DMX512 created 1986 revised 1990**
- **Developed by United States Institute for Theatrical Technology (USITT)**
- **Entertainment Services and Technology Association (ESTA) took over in 1998**
- **DMX512A - 1998 revised 2008 (ESTA)**
- **ANSI Standard since 2004 (Current Revision E1.11-2008)**

DMX512
Introduction

- Simple serial protocol
- Uses RS485 electrical layer
- 250K Baud data rate
- NO ERROR CHECKING
- 512 bytes of data (512 Channels = 1 Universe)
- Multiple Universes allowed
- Single master daisy chain type network
- 5 Pin XLR connectors.
- Special shielded data cable requirements.
- RJ45 and CAT5E for permanent fixtures (DMX512A).

DMX512
Introduction

- **DMX Terminology**
  - Controller = Device sending the DMX data
  - Receiver = Device receiving the DMX data
  - Fixture = Light / Scanner / Fogger etc

MICROCHIP
MASTERs 2013

DMX512
Introduction

- **Common Uses**
  - Dimming Lights
  - Fog Machines
  - Color Mixing Fixtures
  - Robo Lights
  - Scanners
  - CYC Lights
  - House lights
  - Strobes
  - Much More

DMX512
Introduction

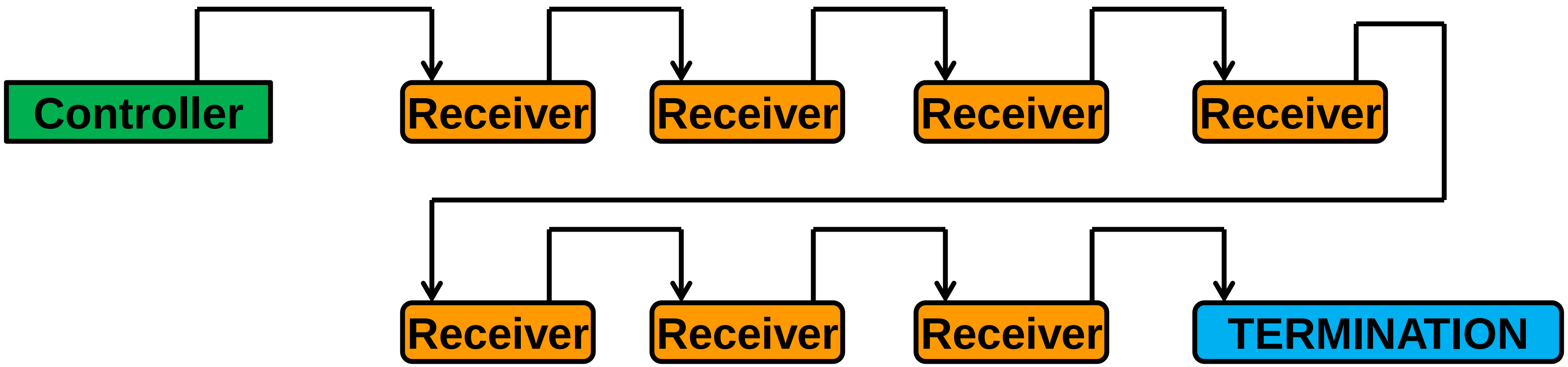
- **Prohibited Uses**
  - Use in anything that could compromise human or animal safety is **PROHIBITED!**

  - Moving Sets………… **NOT ALLOWED!**
  - Pyrotechnic Control… **NOT ALLOWED!**
  - Truss Motion Control.. **NOT ALLOWED!**

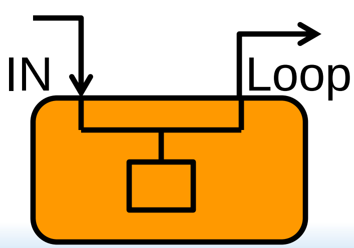
  - All because there is no error checking!

DMX512
Physical Layer

- **Connection Topology**



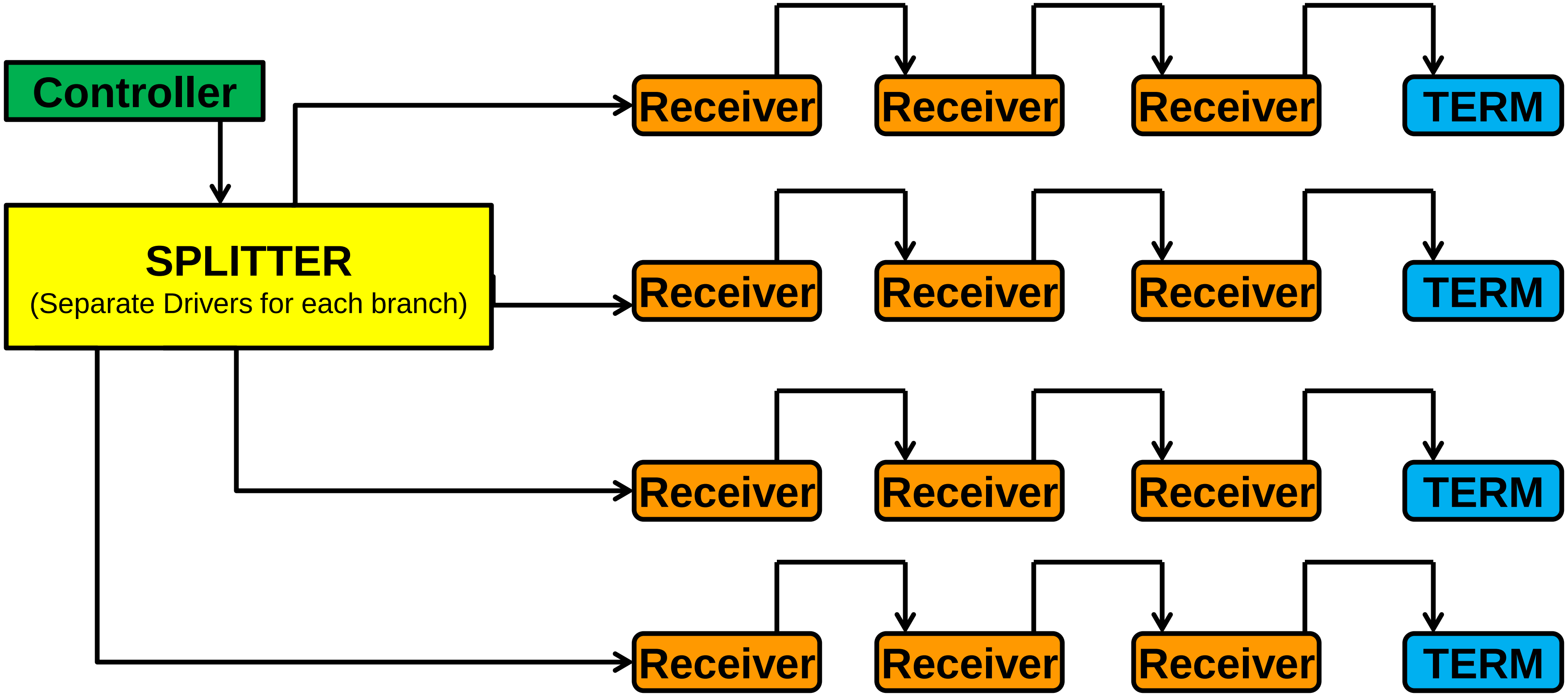
Up to 32 Daisy Chained devices with a 120ohm termination resistor at far end of the chain.



The drawing above shows how each unit connects together. Physically the data bus is a continuous connection using “In” and “loop” connectors wired internally as shown in the block on the left.

DMX512
Physical Layer

- **Connection Topology**



Each output from the splitter can have 32 devices.

DMX512
Physical Layer

- **Electrical**
  - Uses RS485 transceivers
  - Differential Data Transmission
  - Max 32 devices per bus (Excluding Master)
  - Max 1200m (3900 feet)
  - Driver output range +/-1.5V to +/-6V
  - Receiver sensitivity +/-200mV

DMX512
Physical Layer

Connectors

- **5 Pin XLR Plugs and Sockets are only connector specified by DMX512 standard**
- **RJ45 Added in DMX512A for permanent Installation**

DMX512
Physical Layer

Pin Outs

XLR-5 pin out

1. Signal Common
2. Data 1- (Primary Data Link)
3. Data 1+ (Primary Data Link)
4. Data 2- (Optional Secondary Data Link)
5. Data 2+ (Optional Secondary Data Link)

PCB Plug = In

PCB Socket = Loop

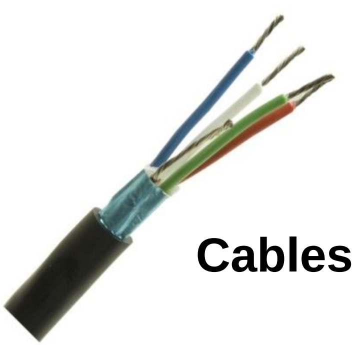
RJ-45 pin out

1. Data 1+
2. Data 1-
3. Data 2+
4. Not Assigned
5. Not Assigned
6. Data 2-
7. Signal Common (0 V) for Data 1
8. Signal Common (0 V) for Data 2

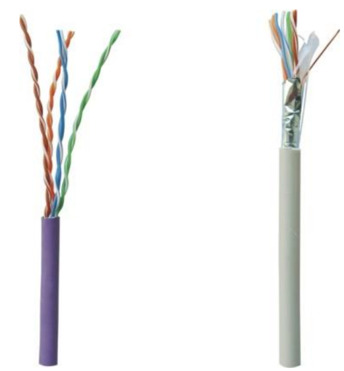
DMX512
Physical Layer

DMX512 CABLE REQUIREMENTS

- **Nominal characteristic impedance of 120 ohms**
- **Two twisted pairs (only 1 pair used)**
- **Shielded**
- **Exception is for permanent installation where CAT5E can be used (DMX512A).**



Cables with DMX512 rating



CAT5E can be used for DMX512A Permanent Installation

MASTERs 2013

DMX512
Protocol

Data Packet

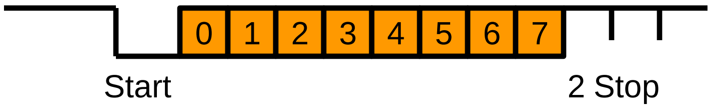
BREAK – Indicates start of new packet – Line low for > 92us (176us Typical)

MAB – Mark After Break - Line High for >12us and <1sec

START – 8 bit Start Code indicates data type – 0x00 for dimmer/general data

Data 1 to Data 512 – 8 bit data sent to devices – up to 512 data slots

Data bytes are 1 start bit, 8 data bits, 2 stop bits at rate of 250kbits/s, LSB first



DMX512
Protocol

Timing

Signal Name	Transmit Timing	Receive Timing
BIT RATE	250kbits/sec	250kbits/sec
BREAK	>92us Typical 176us	>88us
MAB	>12us and <1sec	>8us
Mark Before Break	0 to <1sec	0 to <1sec
Break to Break	1204us to 1sec	1196us to 1.25sec
DMX512 Packet	1204us to 1 sec	1196us to 1.25sec

See ANSI E1.11-2008 Specification for complete timing details

DMX512
Protocol

Addressing

- Strictly Sequential
- Address 1 is the Data byte following the START
- Auto incrementing
- Maximum of 512 data slots (One DMX Universe)

DMX512
Protocol

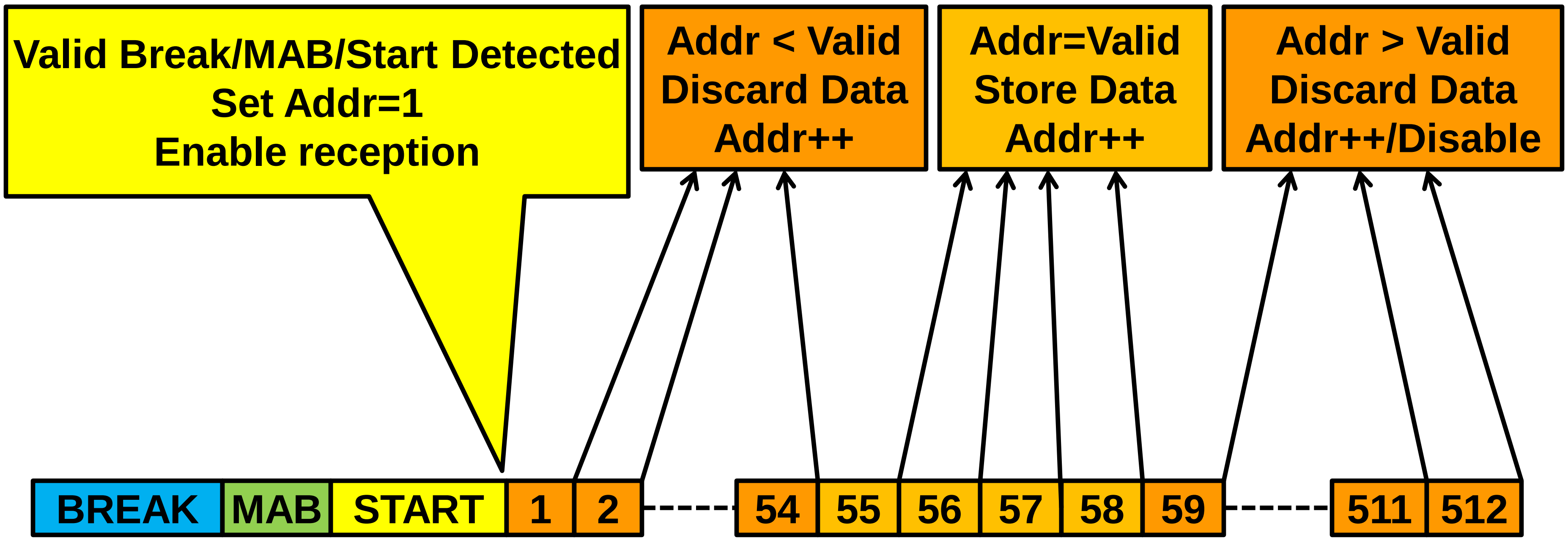
Data Decoding

- Wait for valid BREAK, MAB and START
- If valid then clear and start address counter
- Read individual data bytes as they arrive
- If address counter matches then store the data
- Increment address counter after each data byte
- When address counter >512 stop counter

DMX512
Protocol

Data & Address Decoding Example

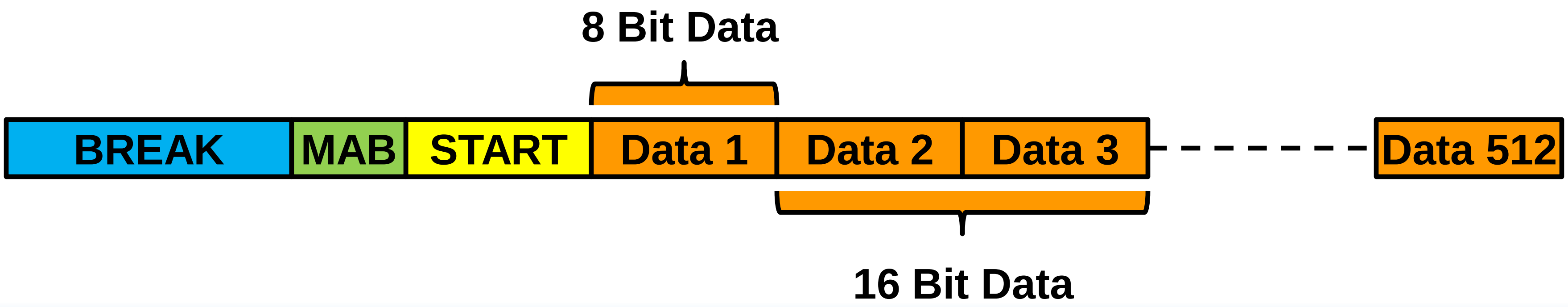
- Our Receiver has a base address of 55
- Our Receiver uses 4 channels of data
- i.e. **Valid** Address Range is address 55 to 58



DMX512
Protocol

8 vs 16bit data

- 16 bit is **NOT** defined in DMX512 specification
- All data is 8 bit all of the time!
- 16 bit data is just two 8 bit bytes combined
- Up to your device to decode data as required

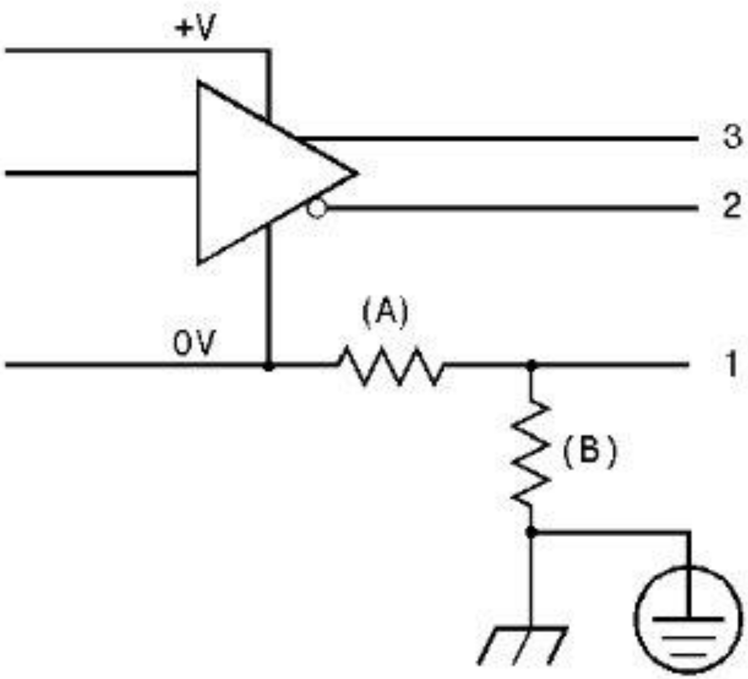


DMX512
Commissioning

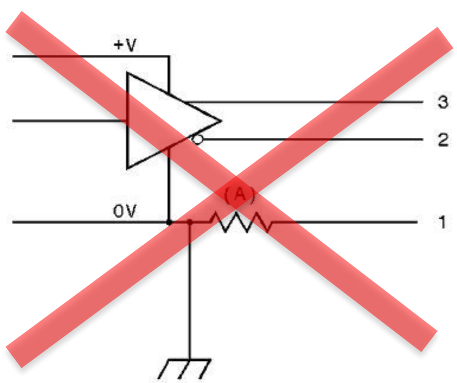
- **Fixed permanent addresses**
- **Set address on Receiver by**
  - Dip Switches
  - Buttons and LED/LCD display*
  - USB set up*
  - **Need to store address in FLASH/EEPROM*
- **Strictly a manual process**

DMX512
Hardware

- **Transmit Circuit**



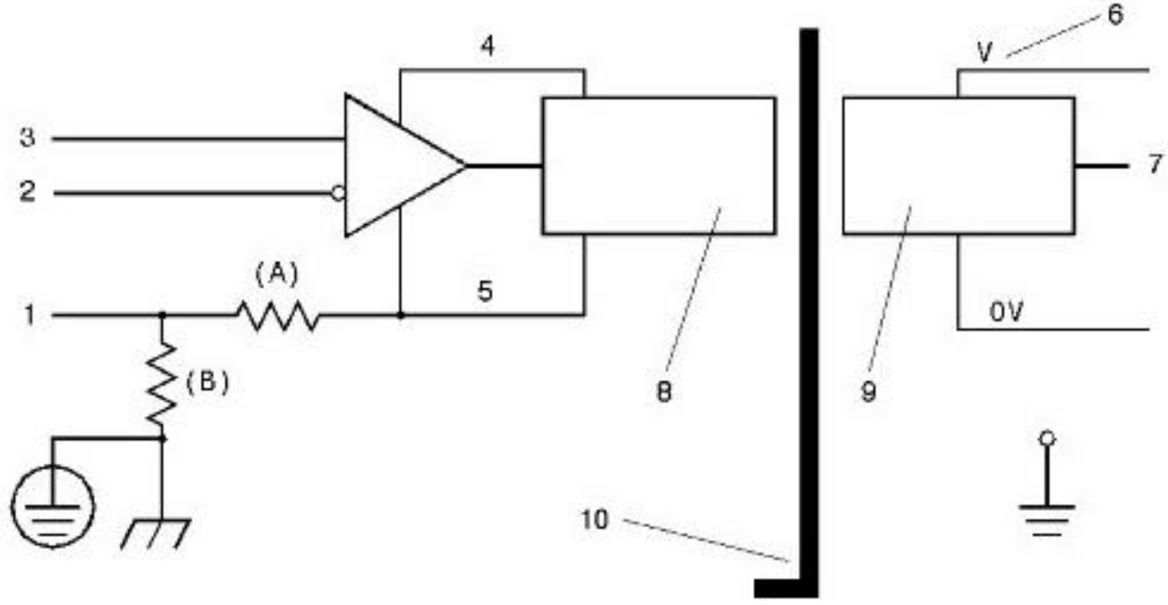
1 – DMX Data Link Common
2 – DMX Data 1+ (or 2+)
3 – DMX Data 1- (or 2-)
A – Optional resistance
B – Optional resistance



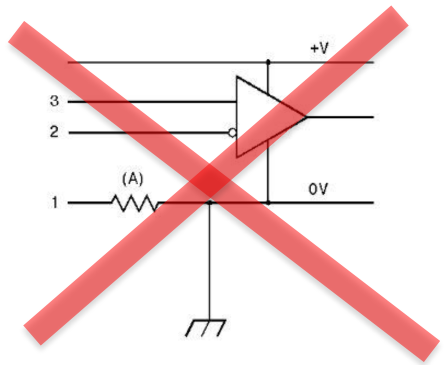
See ANSI E1.11-2008 Specification for circuit and electrical requirements

DMX512
Hardware

- **Receive Circuit**



1 – DMX Data Link Common
2 – DMX Data 1+ (or 2+)
3 – DMX Data 1- (or 2-)
4 – Isolated Supply
5 – Isolated 0V supply
6 – V+
7 – I/O
8 – Isolated Electronics
9 – Optional non Isolated Electronics
10 – Isolation Barrier
A – Optional resistance
B – Optional resistance



Examples of isolated RS485 Transceivers:
ADM2582EBRWZ, MAX3535, IL485, MAX1490

See ANSI E1.11-2008 Specification for circuit and electrical requirements

DMX512
Library

- **DMX512 Library**
  - Controller and Receiver in one library
  - Uses one EUSART
  - Uses one Timer
  - Interrupt Driven
  - Simple API to make it easy to use
  - Some setup required in header file to define pins and EUSART used

DMX512
Library

- **COMMON DMX APIs**
  - `dmx_init()`
  - `dmx_interrupt()`
  - `dmx_set_start(start)`

`Timing:`

  - `dmx_timer_ms()`
  - `dmx_ms_count()`
  - `dmx_ms_clear()`
  - `dmx_timer_sec()`
  - `dmx_sec_count()`
  - `dmx_sec_clear()`

DMX512
Library

COMMON DMX APIs

`void dmx_init(void)`

Initializes the DMX512 library using the settings in dmxconfig.h

Must be called near the beginning of main()

`void dmx_interrupt(void)`

The interrupt function that runs the DMX512 library

Must be called from your interrupt service routine.

DMX512
Library

COMMON DMX APIs

`void dmx_set_start(uint8_t start)`

Sets the DMX512 command byte value
For DMX Controller, this is the start code transmitted
For DMX Receiver, this is the start code to respond to
Library defaults to 0.

DMX512
Library

- **CONTROLLER API**
  - `dmx_write_byte(addr,data)`
  - `dmx_write(addr,*data,num)`
  - `dmx_read_byte(addr)`
  - `dmx_tx_enable(enable)`
  - `dmx_tx_done()`
  - `dmx_get_address()`

DMX512
Library

```
void dmx_write_byte(uint16_t addr,
                    uint8_t data)
```

Writes a data byte to the DMX512 address. Clamps address to match the buffer size set in dmxconfig.h.

```
void dmx_read_byte(uint16_t addr)
```

Reads a single byte from a specific address in TX buffer.

DMX512
Library

`void dmx_tx_done(void)`

Indicates when a packet has been sent.
Self clears

`void dmx_tx_enable(uint8_t enable)`

Enables or disables a transmission sequence meaning turns the transmitter on/off

`void dmx_tx_get_enable(void)`

Returns 1 if transmitter enabled else returns 0

DMX512
Library

```
void dmx_write (uint16_t addr,
                              uint8_t *data,
                               uint8_t num)
```

- Copies an array of data to the output buffer
- addr is the buffer to start writing to (1 to DMX_TX_BUFFER_SIZE, 0 is invalid)
- *data is the pointer to the data to be copied
- num is the number of bytes to copy

MICROCHIP
MASTERs 2013

DMX512
Library

- **CONTROLLER SETUP**
  - dmxconfig.h
    - Select controller mode
    - Select the EUSART to use
    - Select the pin options for BREAK *
    - Set the buffer size needed
    - Select timer to use
    - Set up timing values

*Some older EUSARTS do not allow individual pin control, in these cases separate pin needs to be used to generate the BREAK signal for DMX512.

DMX512
Library

Simple examples using the Controller API

```
#include "dmx.h"
void main (void)
{
  dmx_init();
  adc_init();
  PIE=1;
  GIE=1;
  dmx_set_start(0);
  while(1)
  {
    DelayMs(50);
    dmx_write_byte(1,ReadADC(1));
    dmx_write_byte(10,ReadADC(2));
    dmx_write_byte(252,ReadADC(3));
  }
}
void interrupt MyISR(void)
{
  dmx_interrupt();
}
```

```
#include "dmx.h"
void main (void)
{
  uint8_t data[4];
  dmx_init();
  adc_init()
  PIE=1;GIE=1;
  dmx_set_start(0);
  while(1)
  {
      DelayMs(50);
      data[0]=ReadADC(1);
      data[1]=ReadADC(2);
      data[2]=ReadADC(3);
      data[3]=ReadADC(4);
      dmx_write(1,data,4); //Atomic
  }
}
void interrupt MyISR(void)
{
  dmx_interrupt();
}
```

DMX512
Library

- **RECEIVER API**
  - `dmx_rx_get_start()`
  - `dmx_new_data()`
  - `dmx_read_byte(offset)`
  - `dmx_read(offset,*data, num)`
  - `dmx_set_address(base)`
  - `dmx_get_address()`
  - `dmx_rx_timeout()`

DMX512
Library

`void dmx_rx_get_start(void)`

Returns DMX512A start code in use

`void dmx_set_address(uint16_t base)`

Sets the base address of this receiver.
Data will be read starting from this address.
The amount of data to be read is set in dmxconfig.h by setting the receive buffer size.

DMX512 Library

```
uint8_t dmx_new_data(void)
```

Returns 1 when new data packet has been received. This only responds after the required number of bytes has been read starting at the base address.

```
uint8_t dmx_read_byte(uint8_t
                          offset)
```

Returns a data byte from the receive buffer at the position set by **`offset`**.

DMX512
Library

```
void dmx_read( uint8_t offset,
               uint8_t *data,
               uint8_t num)
```

Copies a block from the receive buffer to ***data** starting at **offset** for a count of **num**.

The receive buffer size set in dmxconfig.h limits how much data can be read.

DMX512
Library

```
uint8_t dmx_set_address(uint16_t
                                base)
```

Sets the base address to start reading data from

```
uint8_t dmx_get_address(void)
```

Gets the base address to start reading data from

```
uint8_t dmx_rx_timeout(void)
```

Returns 1 if 1sec time out occurs

DMX512
Library

- **RECEIVER SETUP**
  - dmxconfig.h
    - Select receiver mode
    - Select the EUSART to use
    - Set the buffer size needed
    - Select timer to use
    - Set up timing values

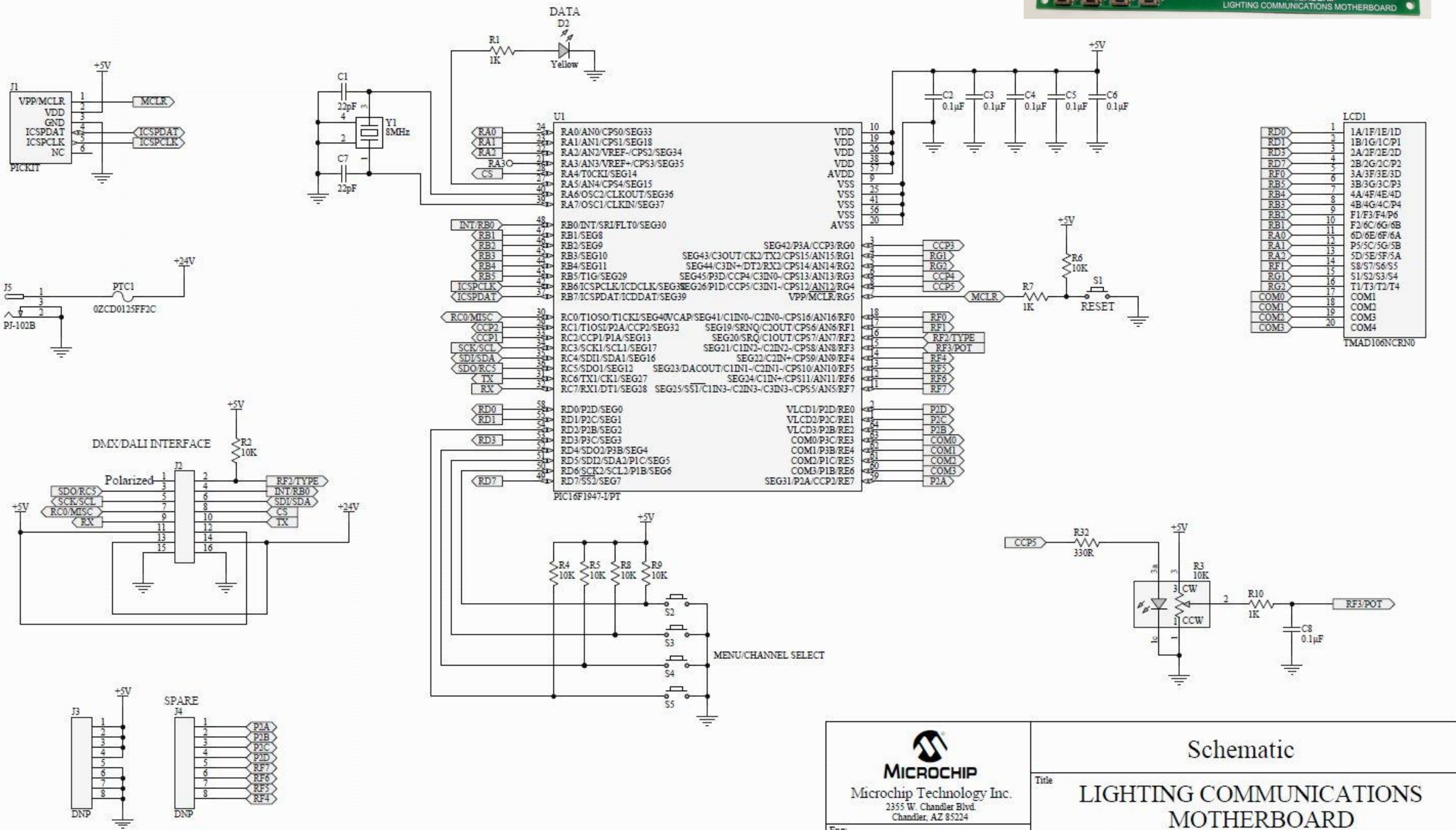
DMX512
Library

Simple examples using the Receiver API

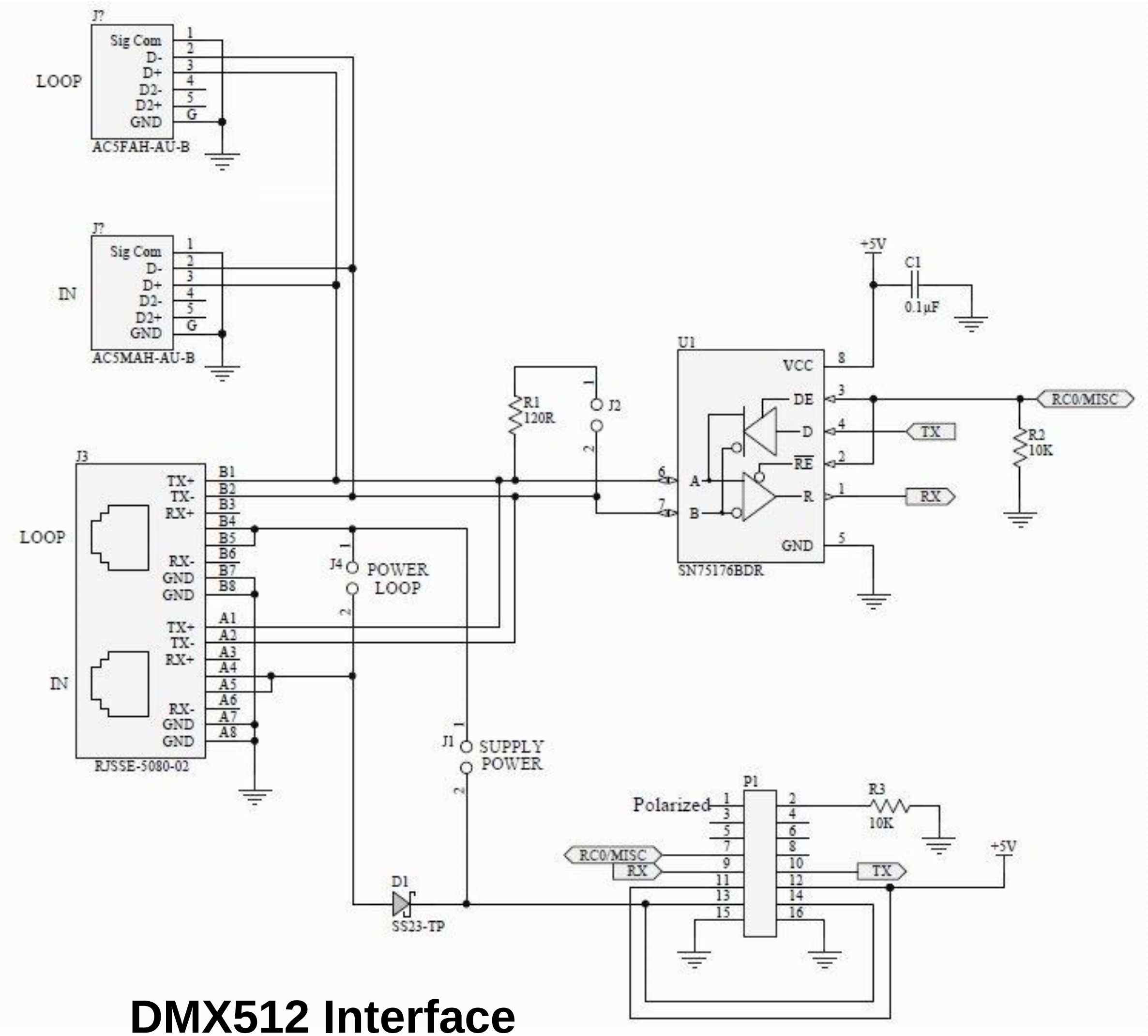
```
#include "dmx.h"
void main (void)
{
  dmx_init();
  led_init();
  PIE=1;
  GIE=1;
  dmx_set_address(base);
  while(1)
  {
    if(dmx_new_data())
    {
      led_set(0,dmx_read_byte(0));
      led_set(3,dmx_read_byte(3));
    }
  }
}


void interrupt MyISR(void)
{
  dmx_interrupt();
}
```

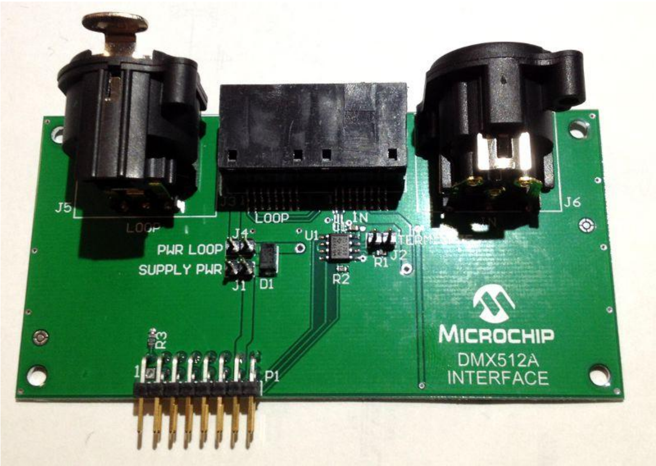
```
#include "dmx.h"
void main (void)
{
  uint8_t data[3];
  dmx_init();
  led_init();
  PIE=1;GIE=1;
  dmx_set_address(base);
  while(1)
  {
    if(dmx_new_data())
    {
      dmx_read(0,data,3);
      led_set(0,data[0]);
      led_set(1,data[1]);
      led_set(2,data[2]);
    }
  }
}
void interrupt MyISR(void)
{
  dmx_interrupt();
}
```

DMX512
Lab Hardware



DMX512
Lab Hardware





DMX512 Interface

Agenda

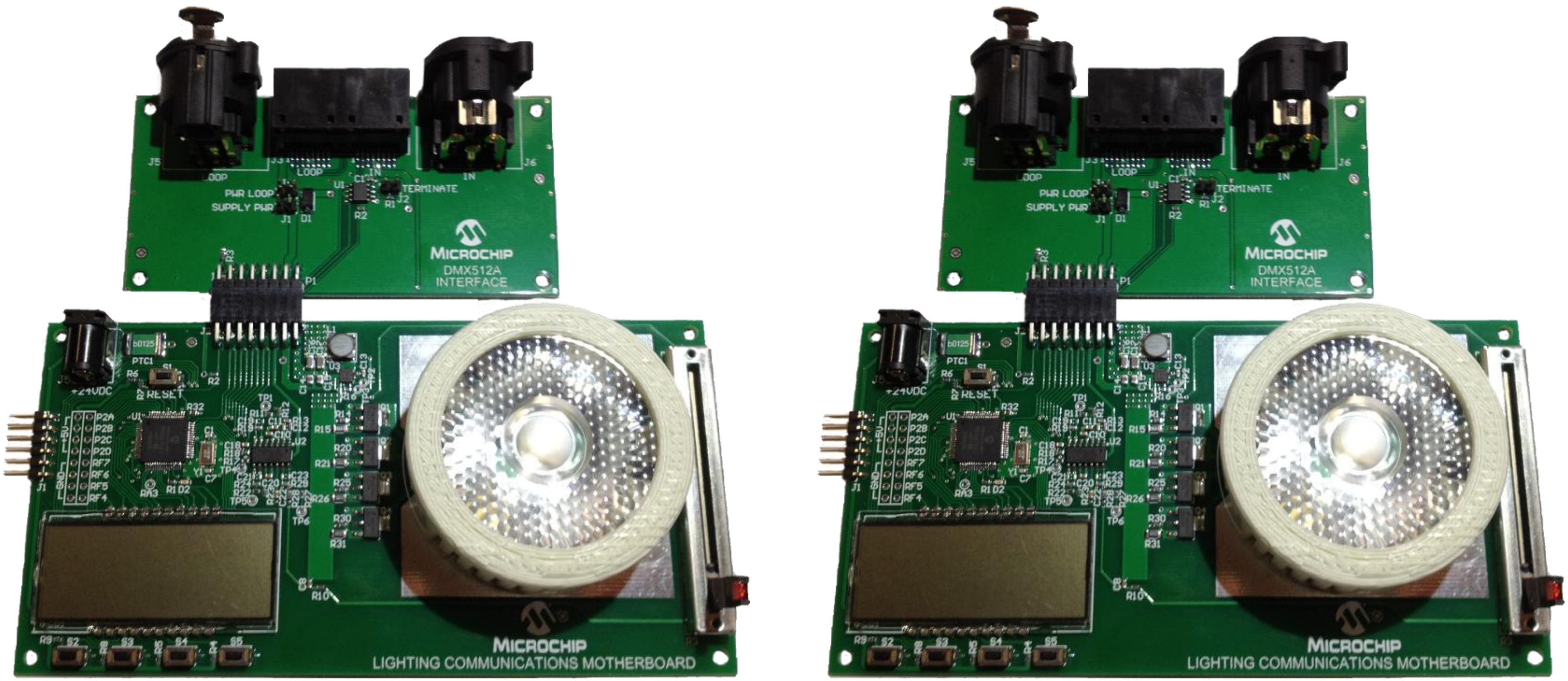
- Lighting Protocols Overview
- DMX512
- **Lab 1 - DMX512 Controller**
- Lab 2 - DMX512 Receiver
- DALI
- Lab 3 - DALI Control Device
- Lab 4 - DALI Control Gear
- Large Lighting Network
- Lab 5 - Implementing a Larger Lighting Network
- Summary

Lab 1:
DMX512 Controller

Lab 1 Objectives

- Set up Library for Controller
- Set up the interrupt
- Set up Controller control value
- Send DMX Data

Lab 1


MICROCHIP
DMX512A
INTERFACE
MICROCHIP
LIGHTING COMMUNICATIONS MOTHERBOARD
MICROCHIP
DMX512A
INTERFACE
MICROCHIP
LIGHTING COMMUNICATIONS MOTHERBOARD

Lab 1 Summary

- Easy to set up hardware
- Easy to add to Interrupt routine
- Easy to send DMX data

Agenda

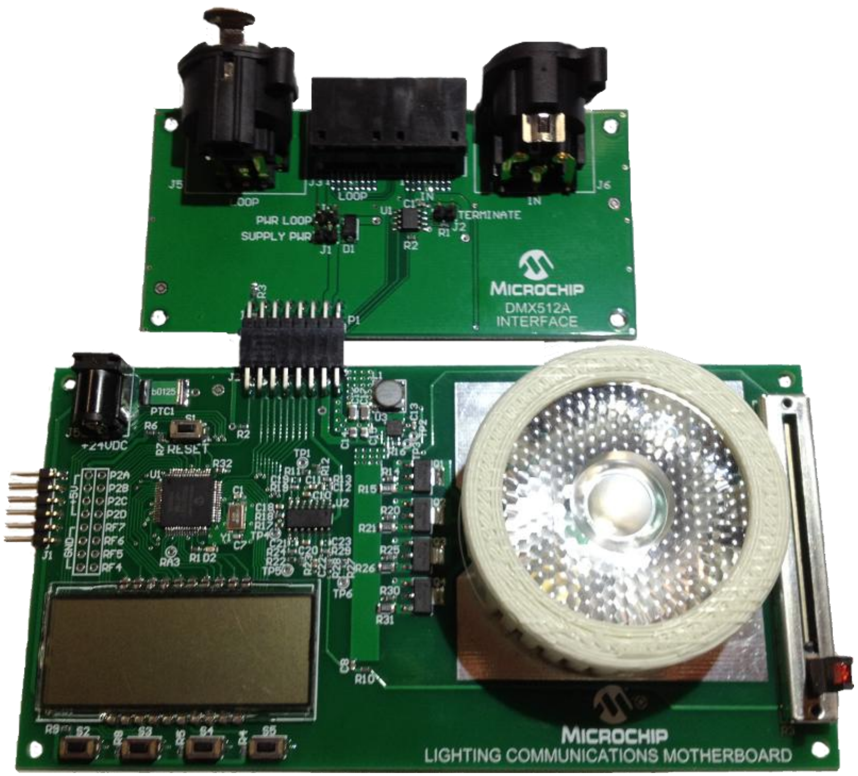
- Lighting Protocols Overview
- DMX512
- Lab 1 - DMX512 Controller
- **Lab 2 - DMX512 Receiver**
- DALI
- Lab 3 - DALI Control Device
- Lab 4 - DALI Control Gear
- Large Lighting Network
- Lab 5 - Implementing a Larger Lighting Network
- Summary

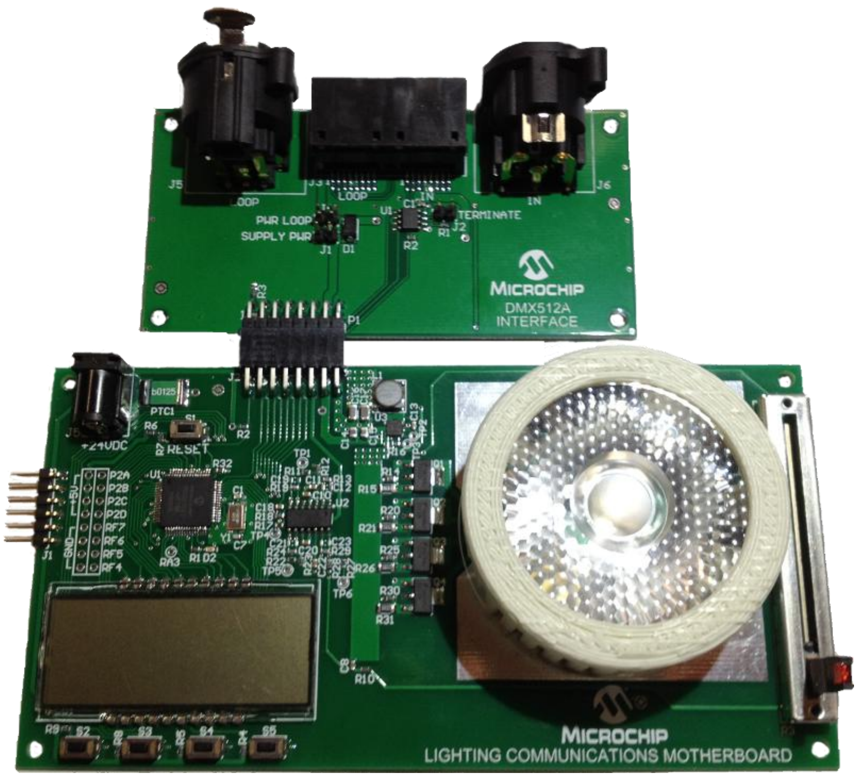
Lab 2:
DMX512 Receiver

Lab 2 Objectives

- **Set up Library for Receiver**
- **Set up interrupt function**
- **Set up Receiver control value**
- **Read DMX512 Data**

Lab 2


MICROCHIP
DMX512A
INTERFACE
TERMINATE
PWR LOOP
SUPPLY PWR
RESET
MICROCHIP
LIGHTING COMMUNICATIONS MOTHERBOARD


MICROCHIP
DMX512A
INTERFACE
TERMINATE
PWR LOOP
SUPPLY PWR
RESET
MICROCHIP
LIGHTING COMMUNICATIONS MOTHERBOARD

Lab 2 Summary

- Easy to set up hardware
- Easy to add to Interrupt routine
- Easy to receive DMX512 data

Agenda

- Lighting Protocols Overview
- DMX512
- Lab 1 - DMX512 Controller
- Lab 2 - DMX512 Receiver
- **DALI**
- Lab 3 - DALI Control Device
- Lab 4 - DALI Control Gear
- Large Lighting Network
- Lab 5 - Implementing a Larger Lighting Network
- Summary

DALI
Introduction

- DALI – Digital Addressable Lighting Interface
- Commercial / Industrial
- Alternate to 0-10V and PWM methods
- Open standard alternate to Digital Signal Interface (DSI)
- Part of IEC 60929 Spec for fluorescent ballasts
- Commercial development started around 1998

DALI
Introduction

- Two wire serial bus
- Special DALI power supply required
- 16/8 bit Manchester encoded data packets
- 64 channels / addresses
- 16 groups
- Limited bidirectional communications
- Multi master is allowed
- Variable addressing available
- No Error Checking

DALI
Physical Layer

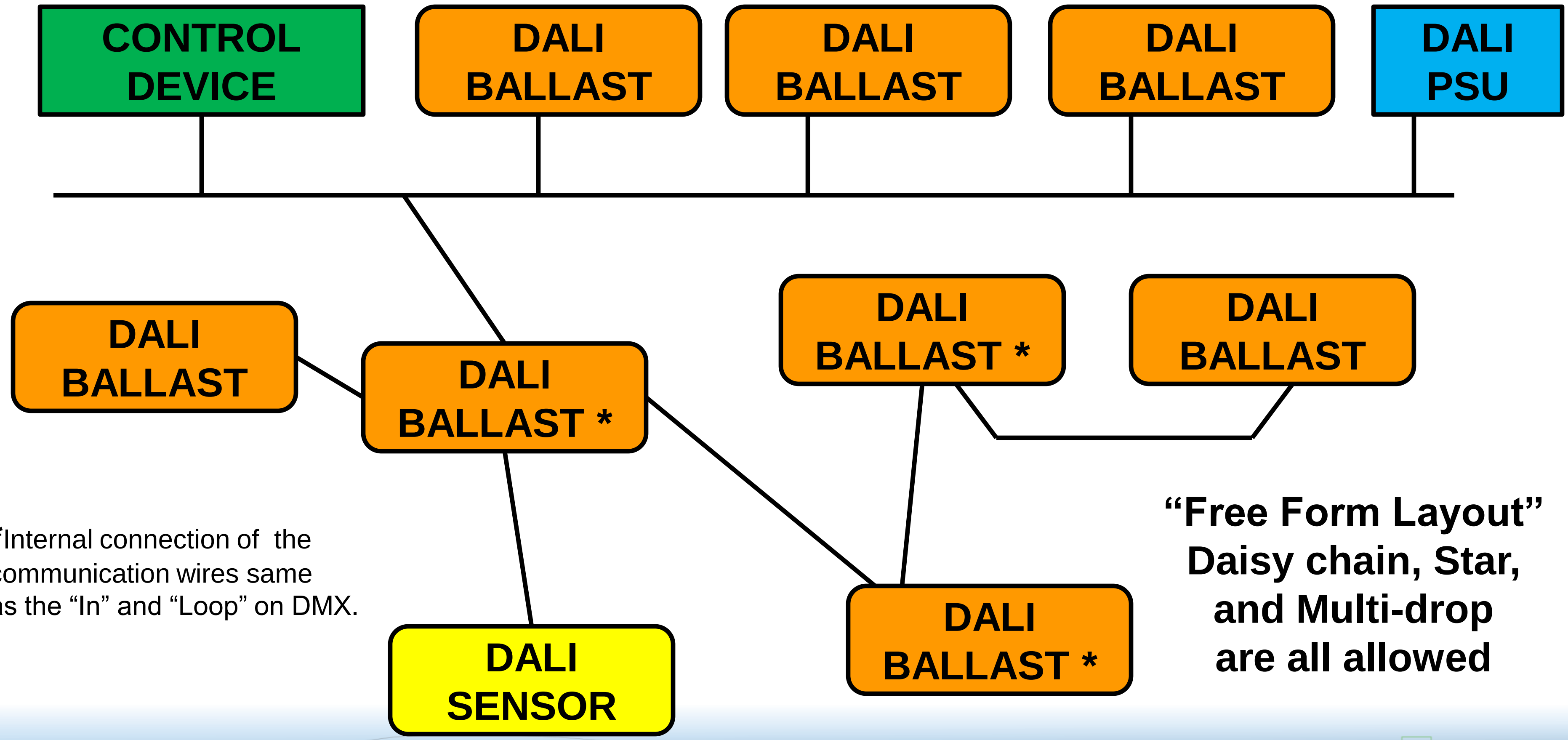
- **DALI Terminology**
  - "**Control Gear**" = Ballast or Sensor (Receiver)

  - "**Control Device**" = Controller (Transmitter)

  - "**Forward Frame**" = Packet sent from Control Device to Control Gear
  - "**Back Frame**" = Response to a Forward Frame from Control Gear

DALI
Physical Layer

- **Connection Topology**



*Internal connection of the communication wires same as the "In" and "Loop" on DMX.

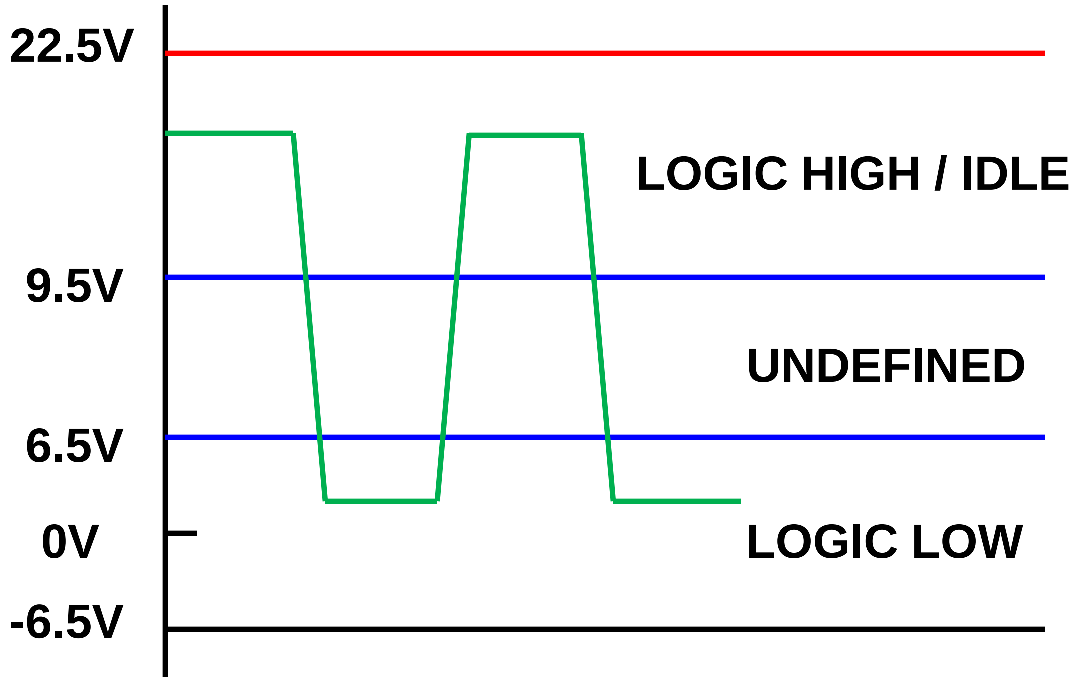
"Free Form Layout" Daisy chain, Star, and Multi-drop are all allowed

DALI
Physical Layer

- **Electrical**
  - DALI PSU 9.5V to 22.5V, 16V nominal. 250mA current limited, <10us response.
  - Idle is when PSU is above 9.5V
  - Active is voltage below 6.5V (Short PSU!)
  - Not polarized at receivers
  - Usually optically isolated
  - Manchester encoding @ 1200 Baud
  - 2mA allowance per device on the bus

DALI
Physical Layer

- **Electrical**



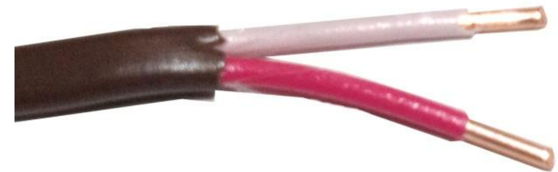
DALI
Physical Layer

- **Connectors**
  - No actual specification
  - Common Screw Terminals or push fit
  - Two wire connector



DALI
Physical Layer

- **Cabling**
  - 2 Wire cable

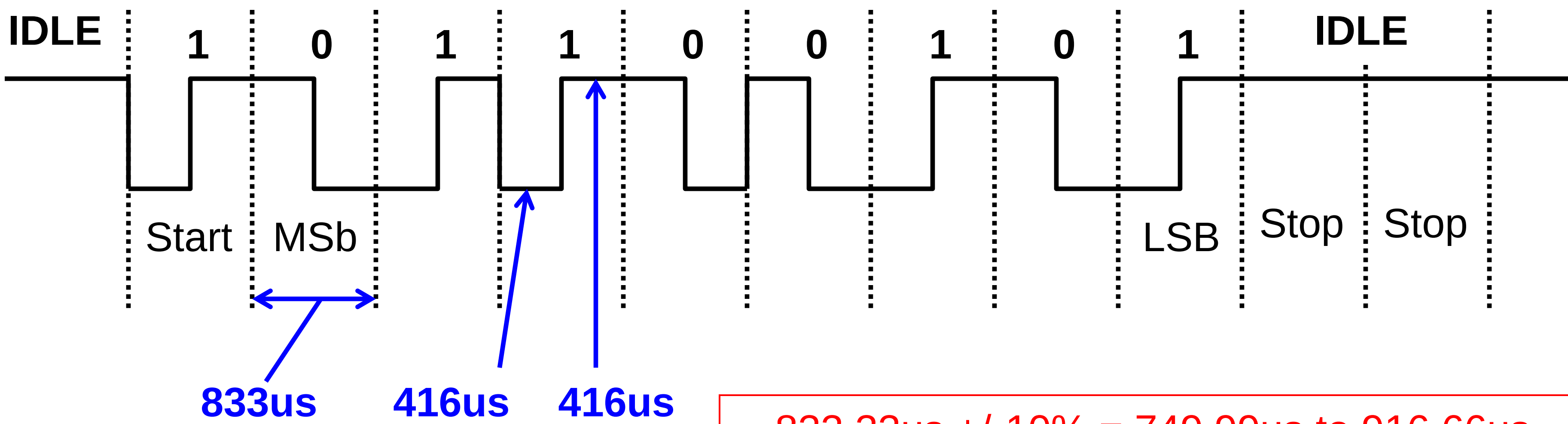


  - No more than 2V drop end to end
  - Standard Electrical cable used
  - 18 AWG common on many fixtures
  - Often Purple in color

  - Class 1 or 2 cable (Solid/Stranded)
  - Usually 600V rated for installation

DALI
Protocol

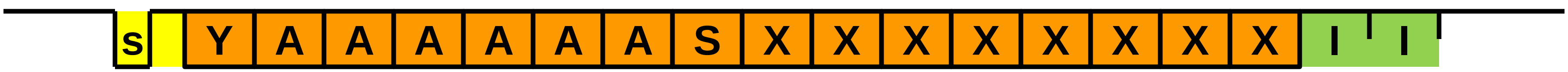
- **Data Transmission**
  - Manchester Encoding – Idle High
  - 1200 Baud +/-10% (833.33us slot)
  - MSb First



833.33us +/-10% = 749.99us to 916.66us
½ Slot timing range of 374.99us to 458.33us

DALI
Protocol

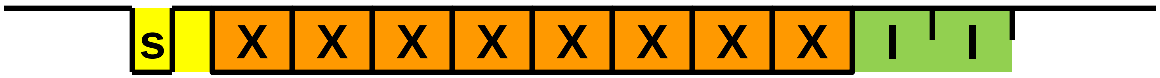
- **Forward Frame**
  - Sent to Ballast, AKA Control Gear
  - 1 Start, 16 Data, 2 Stop, MSB first



S = “1” = Start Bit
Y = “0” = Short Address
Y = “1” = Group Address or Broadcast
A = Address
S = “0” = Direct Arc power following
S = “1” = Command Following
X = 8 bit Direct Arc power or Command
I = Idle = Stop Bits

DALI
Protocol

- **Back Frame**
  - Response from control gear
  - 1 Start, 8 Data, 2 Stop, MSB first



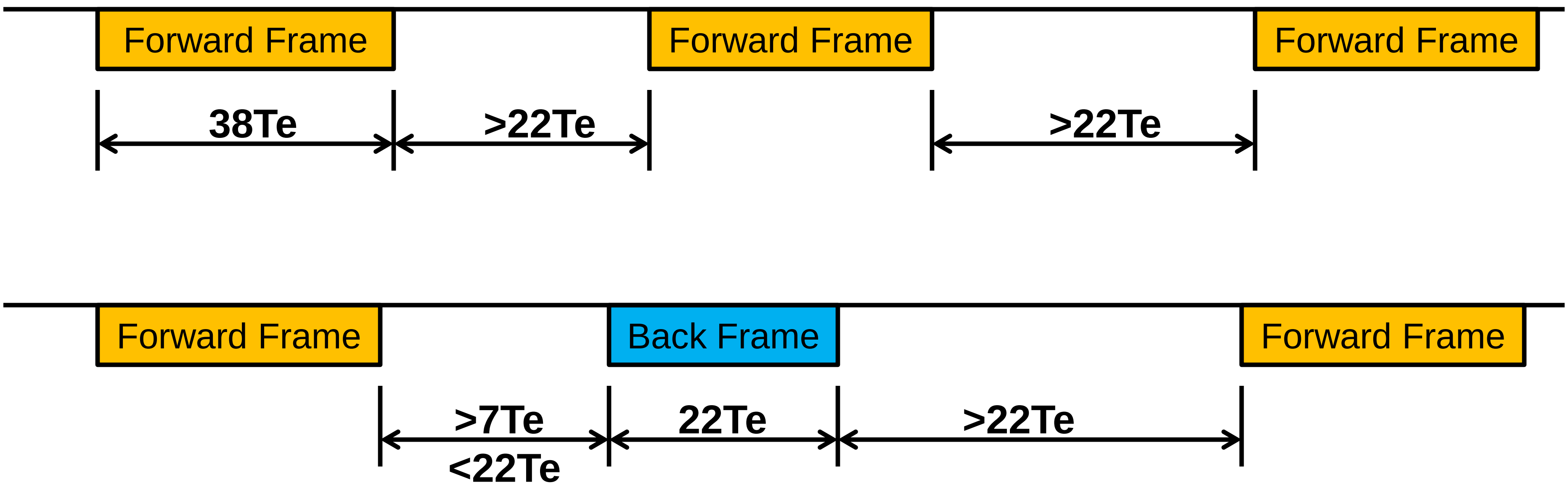
s = “1” = Start Bit
X = Response Data
I = Idle = Stop Bits

A “No” response is no frame sent.
A “Yes” response is 0xFF
Other values vary depending on command that the control gear is responding to.

DALI
Protocol

- **Timing**

Te = Half Bit time
Te= 1/(2 x 1200baud) = 416.67us
38Te = 15.83ms (Forward Frame*)
22Te = 9.17ms (Back Frame*)
7Te = 2.91ms
* Includes Stop Bits

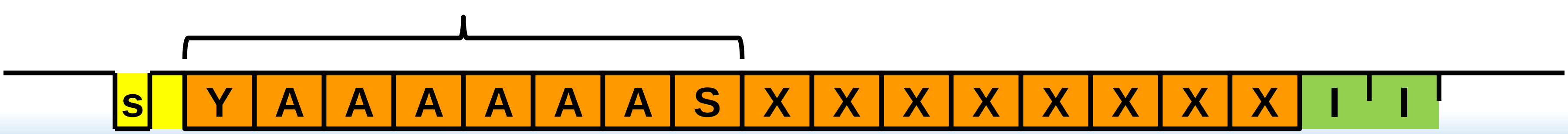


DALI
Protocol

- **Timing**
  - In certain cases a command is repeated within 100ms. This is mentioned in the specific command definitions in the specification.
  - Some commands enable long timeouts (Seconds to minutes) to allow for operations like commissioning. Details in the specification.
  - Answers to broadcast or group addressed queries can overlap resulting in a corrupt back frame.

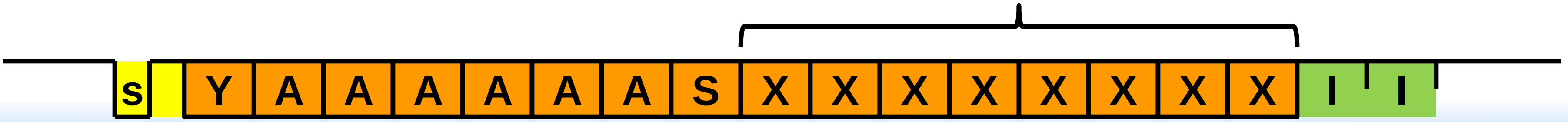
DALI
Protocol

- **Addressing**
  - Short address: `0-63 0AAA AAAS`
  - Group address: `0-15 100A AAAS`
  - Broadcast: `1111 111S`
  - Special commands: `1010 0000`
    `to 1111 1101`
  - S=0 Means the data is a direct Arc level
  - S=1 Means the data byte is a command



DALI
Protocol

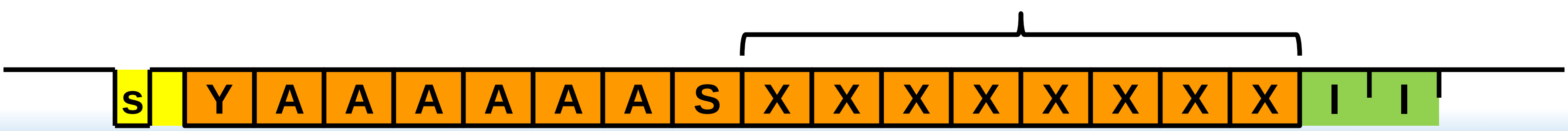
- **Data Decoding**
  - The second byte of the packet is the data.
  - Idle is “NO”
  - 0xFF is “YES”
  - Any other data, is a respond to a query command by the control device



DALI
Protocol

- **Commands**
  - Section 11 of 62386-102 covers these.
  - S=0 is for “Direct Arc Power”
  - S=1 Commands, some listed below

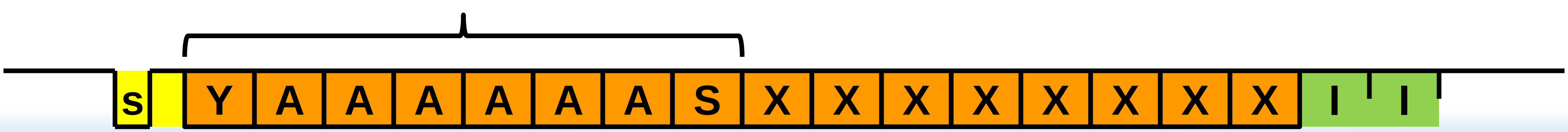
0	OFF	8	ON AND STEP UP
1	UP	9	ENABLE DAPC SEQUENCE
2	DOWN	10-11	Reserved for Future – do not react
3	STEP UP	12-15	Reserved for Future – do not react
4	STEP DOWN	16-31	GO TO SCENE
5	RECALL MAX LEVEL	32	RESET
6	RECALL MIN LEVEL		See the specification for details and
7	STEP DOWN AND OFF		complete list of commands



DALI
Protocol

- **Special Commands**
  - Section 11.4 of 62386-102 covers these.
  - Used for commissioning and extended options
  - List below show a few of the useful commands

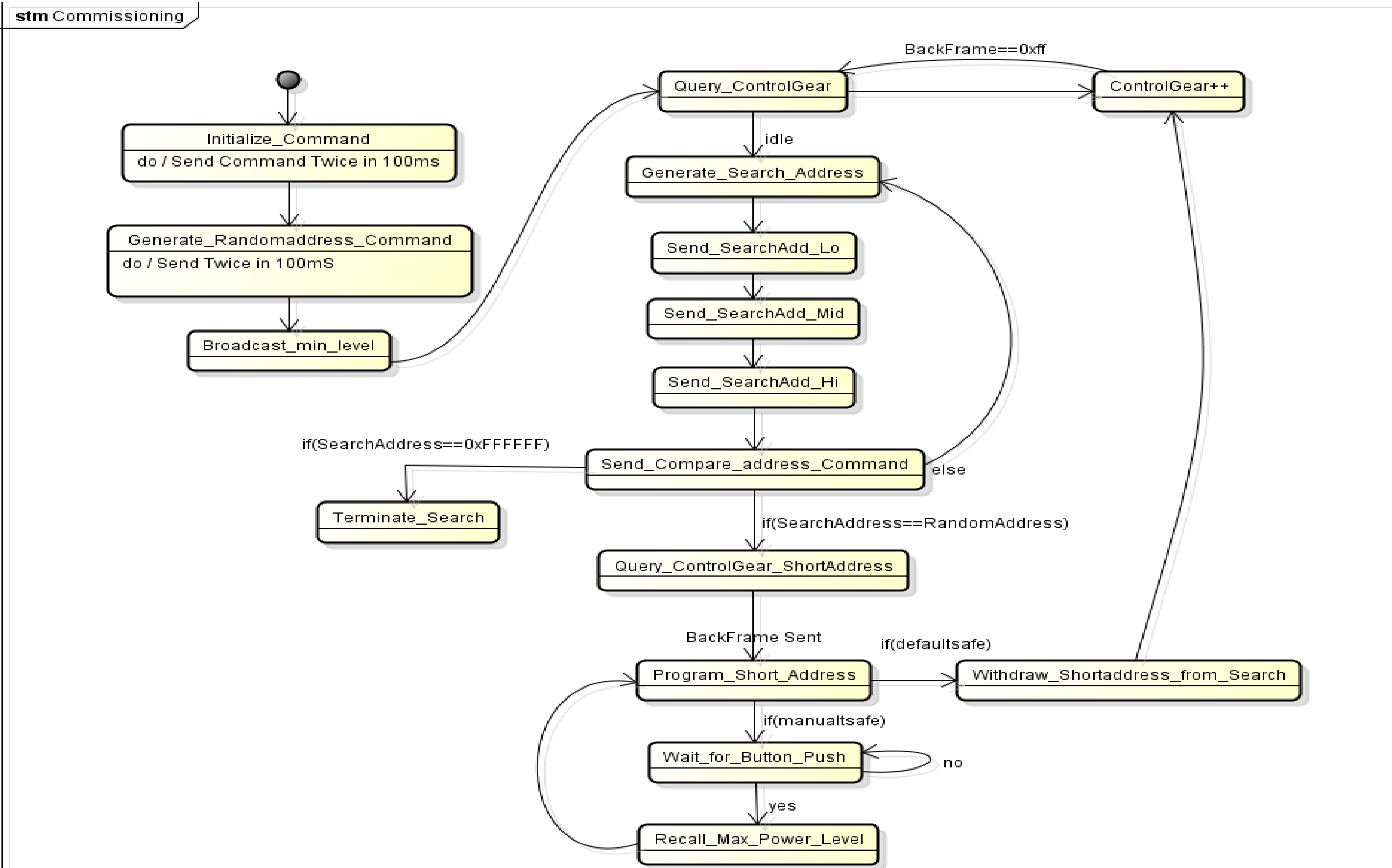
256	TERMINATE	266	SEARCHADDRL
257	DATA TRANSFER REGISTER	267	PROGRAM SHORT ADDRESS
258	INITIALISE	268	VERIFY SHORT ADDRESS
259	RANDOMISE	269	QUERY SHORT ADDRESS
260	COMPARE	270	PHYSICAL SEELCTION
261	WITHDRAW		See specification for details and complete
264	SEARCHADDRH		list of commands.
265	SEARCHADDRM		Including Extended Special Commands



DALI
Commissioning

- **Commissioning sequence**
  - Enter commission mode
  - Set random long address
  - Search long addresses for match
  - Assign short address
  - Repeat search until search is complete
  - End commission mode

DALI
Commissioning

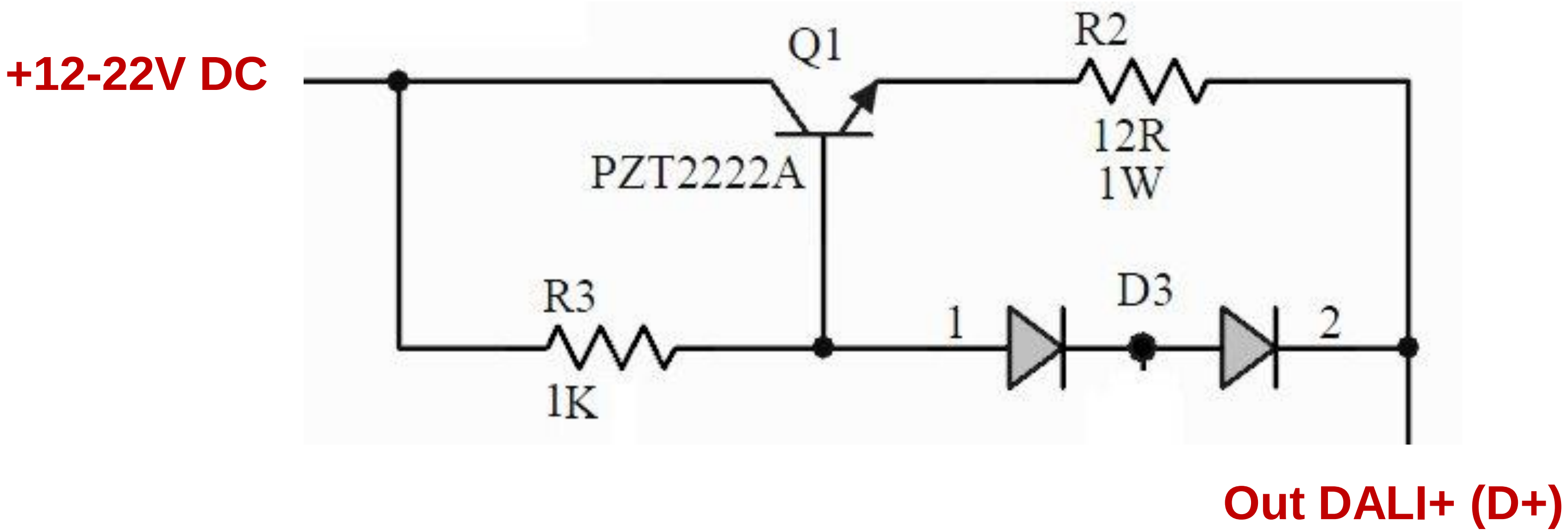


DALI
Hardware

- **DALI Circuits**
  - Specification does not have any circuit designs or recommendations.
  - We have used an isolated method.
  - Also have a simple power supply circuit.

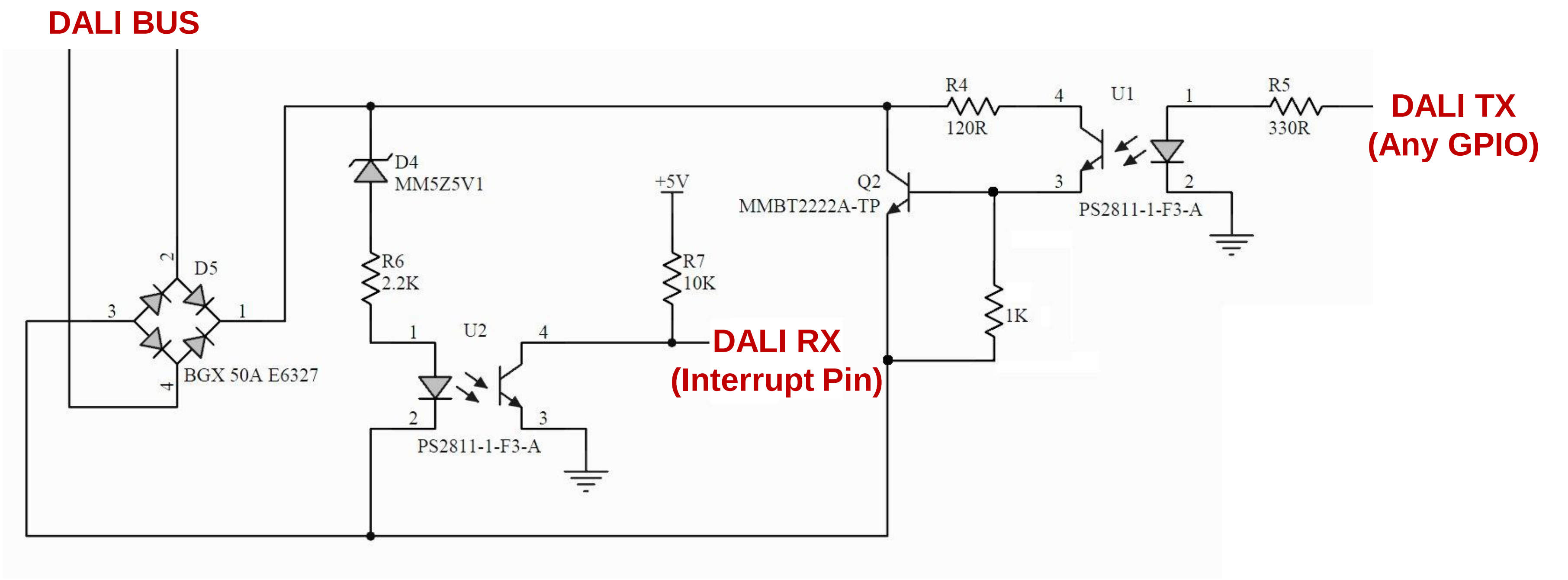
DALI
Hardware

- **DALI Power Supply Circuit**
  - Needs fast response & current limiting.
  - This “simple” circuit can be improved!



DALI
Hardware

- **Isolated Communications Circuit**



DALI
Library

- **DALI Library**
  - Separate Device(Master) and Gear(Slave) libraries.
  - Requires 1 x Interrupt input pin
  - Requires 1 x 16 bit timer for baud generator
  - Requires 1 x 1ms timer for Tick and Time (Shared)
  - Interrupt Driven Transmit and Receive
  - Command processing done in while(1) loop
  - BASIC commands & commissioning only.

DALI
Control Device Library

- Control Device API
  - `DALI_Init()`
  - `DALI_Interrupt()`
  - `DALI_Gear_ShortAddress()`
  - `DALI_Select_Address()`
  - `DALI_Commissioning_Done()`
  - `DALI_Run()`
  - `DALI_Turn_on(Button_Pressed)`
  - `DALI_Turn_off(Button_Pressed)`
  - `DALI_Dim_Up_Down(val,gear)`
  - `DALI_Start_Commissioning()`

The API Library is still under construction

DALI
Control Device Library

`void DALI_Init(void)`

Initializes the DALI Library using the settings as configured in daliconfig.h

Call once at the beginning of your main().

`void DALI_Interrupt(void)`

This is the function that processes the interrupts used to make DALI operate.

This must be called from inside your interrupt service routine.

DALI
Control Device Library

`uint8_t DALI_Gear_ShortAddress(void)`

This function returns the short Address you want to program to the control gear. It can be changed to however user wants to program the control gear address

`uint8_t DALI_Select_Address(void)`

This function returns the short address chosen by the user. This function calls the DALI_Gear_ShortAddress() to get that information and is used during commissioning

DALI
Control Device Library

```
void DALI_Commissioning_Done(void)
```

Returns ‘1’ if true else returns ‘0’

```
void DALI_Run(void)
```

Starts the Transmit process in case of Control Device

```
void DALI_Turn_On(uint8_t
                        Button_Pressed)
```

Sets the DALI turn on Command and takes the address of the button pressed, to turned on the gear at that address

DALI
Control Device Library

```
void DALI_Turn_Off(uint8_t
                    Button_Pressed)
```

Sets the DALI turn off Command and takes the address of the button pressed, to turn gear off at that address

```
void DALI_Dim_Up_Down(uint8_t val,
                      uint8_t gear)
```

'val' is the value you want to change the brightness of the gear to and 'gear' is the address of the gear you want to dim up or dim down

DALI
Control Device Library

`void DALI_Start_Commissioning(void)`

Starts and runs a DALI commissioning sequence (Blocking).

By default this is a fully automated system that just assigns an address in the order they are processed. In an actual end application a user would select a switch to assign an address as each control gear is discovered.

A user callback is available that is called from this function to allow customization of this process, **DALI_Select_Address()**

See the Lab manual and DALI library application notes for more details.

DALI
Control Device Library

- **Dali_Global.h**
  - Select receive interrupt pin / type
  - Select transmit pin
  - Select timers to use
  - Timing calibration

- **Modes**
  - `IDLE` – Waiting for user to initiate a process
  - `TRANSMIT` – Ready to Transmit Commands
  - `RECEIVE` – Waiting to hear back from control gear

DALI
Control Device Library

Simple example using the Control Device API

```
#include "dali.h"
void main (void)
{
  lcm_hw_init();
  DALI_Init();
  while(1)
  {if (ButtonPressed==StartCommissioning || DALI_Commissioning_Done()==CLEAR )
        {DALI_Start_Commissioning(); }
   switch(keypressed())
    {
      case LIGHT1:
         if (status==OFF)
           DALI_Turn_On(LIGHT1);
         else DALI_Turn_OFF(LIGHT1);
         break;
      case default:
        DALI_Dim_Up_Down(level,address); break;
    } } }
void interrupt MyISR(void)
{
  dali_interrupt();
}
```

DALI
Control Gear Library

- **Control Gear Library API**
  - `DALI_Init()`
  - `DALI_Run()`
  - `DALI_GetArcPower()`
  - `DALI_GetArcMin()`
  - `DALI_GetArcMax()`
  - `DALI_Fader()`
  - `DALI_GetArcTable()`
  - `DALI_GetFadeTime()`

The API Library is still under construction

DALI
Control Gear Library

`void DALI_Init(void)`

- Initializes the DALI Library using the settings as configured in daliconfig.h
- Call once at the beginning of your main()

`void DALI_Interrupt(void)`

- This is the function that processes the interrupts used to make DALI operate
- This must be called from inside your interrupt function

DALI
Control Gear Library

`void DALI_Run(void)`

This receives and processes the DALI data sent. It is non-blocking and must be called from your while(1) loop.

`uint8_t DALI_GetArcPower(void)`

Returns the current arc power setting

`uint8_t DALI_GetArcMin(void)`

Returns the minimum dimming value

`uint8_t DALI_GetArcMax(void)`

Returns the maximum dimming value

DALI
Control Gear Library

`uint16_t DALI_Fader(void)`

This function does a lookup from a logarithmic data table to set the PWM duty cycle

`uint16_t DALI_GetArcTable(void)`

Using the Arc power level, this function returns the 16 bit value from the logarithmic dimming table

`Uint8_t DALI_GetFadeTime(void)`

Returns the current fade time

DALI
Control Gear Library

```
uint8_t DALI_Rand(void)
```

DALI requires a unique 24 bit address for commissioning. Because of this a genuine random number generator is needed to avoid collisions.

Since the default rand() function provided with most compilers is only a pseudo random number generator, and given the same seed produces the same output sequence, then a custom random number generator is needed.

This function can be altered by the programmer to use different sources to generate a more realistic random number.

DALI
Control Gear Library

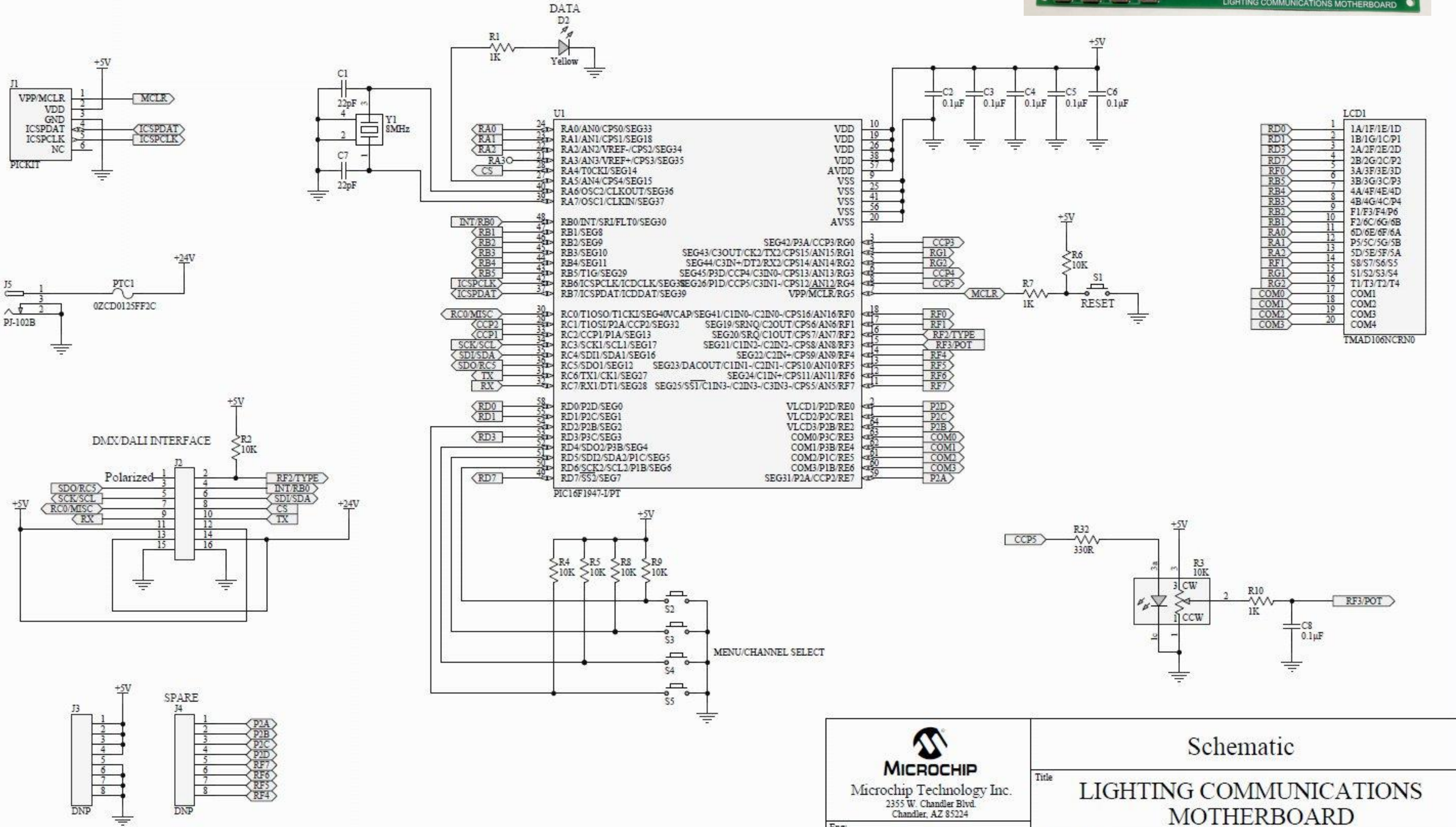
- **Control gear setup**
  - **DALI_Global.h**
    - Select receive interrupt pin / type
    - Select transmit pin
    - Select timer to use
  - **DALI_Commands_Variables.c**
  - Array storing dimming curve data for your lamp to give even brightness steps
    - Fade Rate Table
    - Fade Time Table
    - Log Dimming Table
  - **DALI_Rand.c**
    - The randomizer function for commissioning. This generates a 24bit unique number.

DALI
Control Gear Library

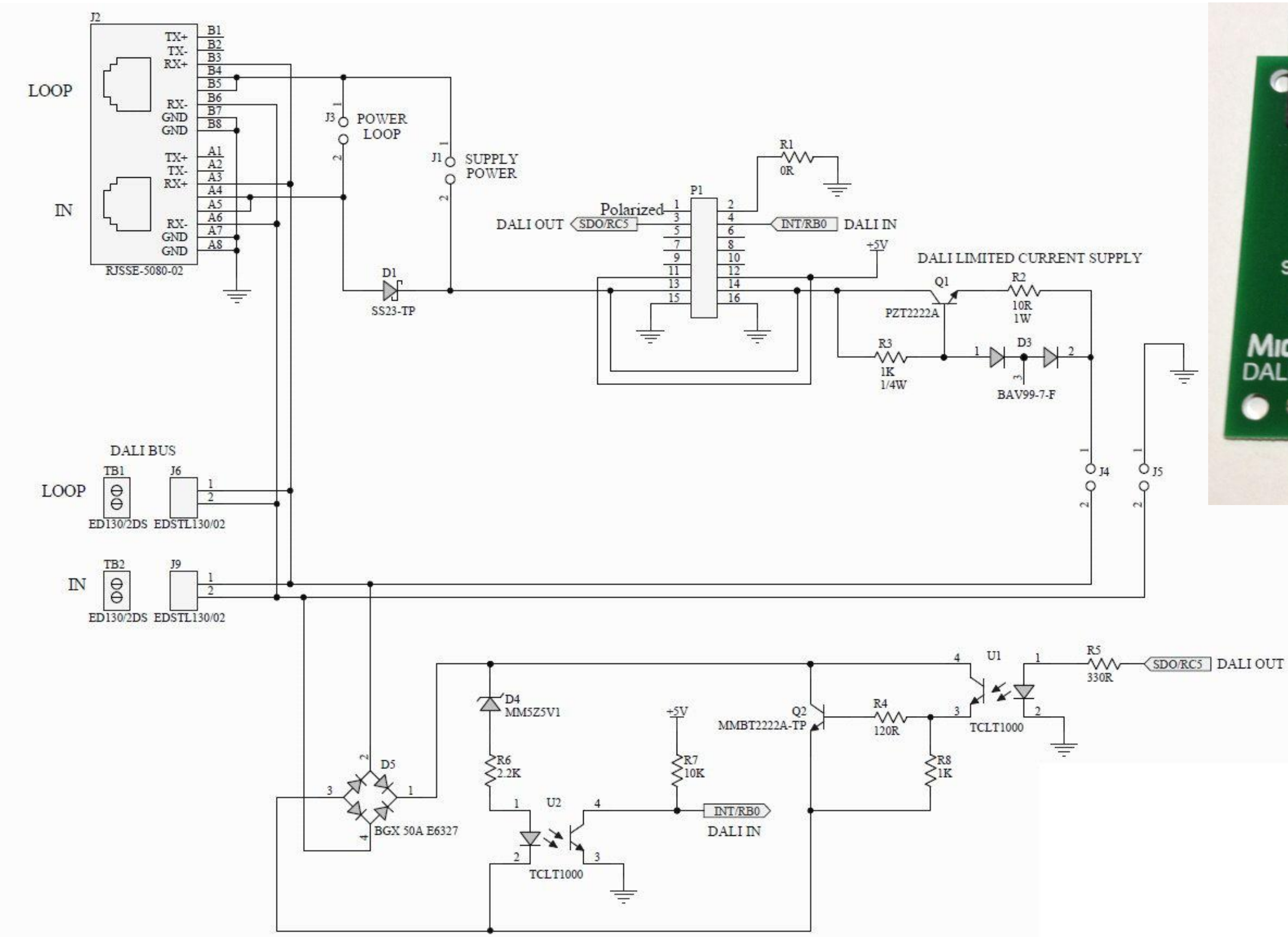
Simple example using the Control Gear API

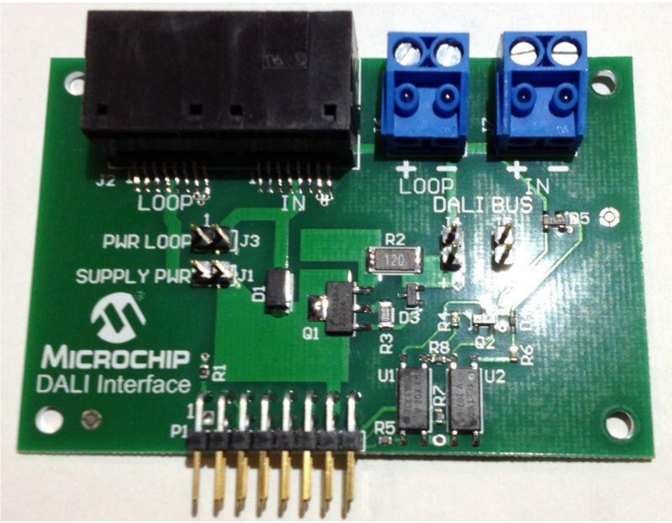
```
#include "dali.h"
void main (void)
{
  lcm_hw_init();
  DALI_Init();
  while(1)
  {
    DALI_Run();
    if(time.tick_10ms)
    {
      time.tick_10ms==0;
      fadecount= DALI_Fader();
      if (fadecount){
        setArcPowerLevel(DALI_GetArcPower();)
      }
    }
  }
}
void interrupt My_Interrupts(void)
{
  DALI_Interrupt();
}
```

DALI
Lab Hardware



DALI
Lab Hardware





DALI Interface

Agenda

- Lighting Protocols Overview
- DMX512
- Lab 1 - DMX512 Controller
- Lab 2 - DMX512 Receiver
- DALI
- **Lab 3 - DALI Control Device**
- Lab 4 - DALI Control Gear
- Large Lighting Network
- Lab 5 - Implementing a Larger Lighting Network
- Summary

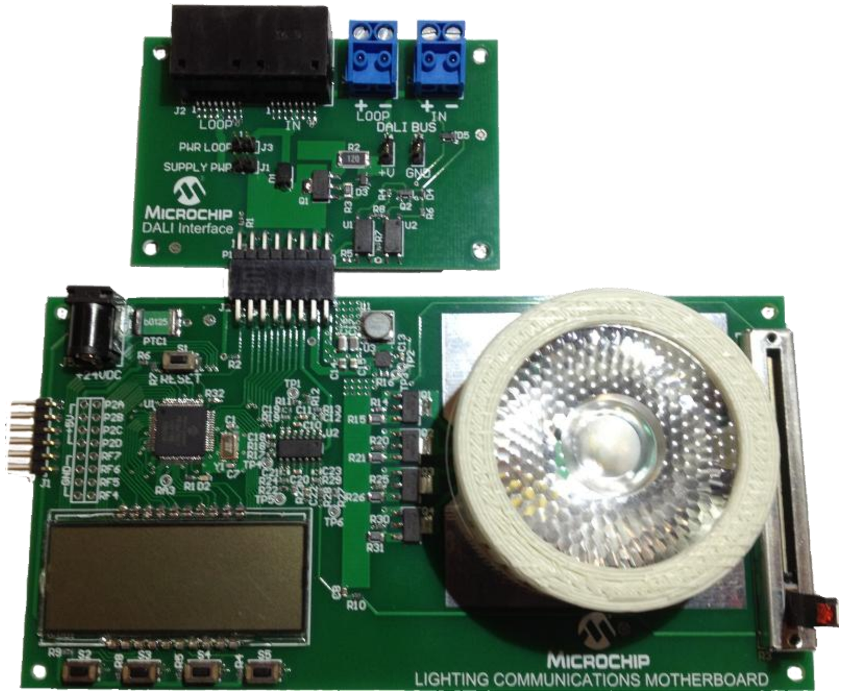
Lab 3:
DALI Control Device

Lab 3 Objectives

- Set up Library for Control Device
- Set up interrupt function
- Send DALI commands
- Do commissioning

Lab 3


MICROCHIP
DALI Interface
MICROCHIP
LIGHTING COMMUNICATIONS MOTHERBOARD


MICROCHIP
DALI Interface
MICROCHIP
LIGHTING COMMUNICATIONS MOTHERBOARD

Lab 3 Summary

- Easy to set up hardware
- Easy to add to Interrupt routine
- Easy to send DALI data
- Easy to do a commissioning

Agenda

- Lighting Protocols Overview
- DMX512
- Lab 1 - DMX512 Controller
- Lab 2 - DMX512 Receiver
- DALI
- Lab 3 - DALI Control Device
- **Lab 4 - DALI Control Gear**
- Large Lighting Network
- Lab 5 - Implementing a Larger Lighting Network
- Summary

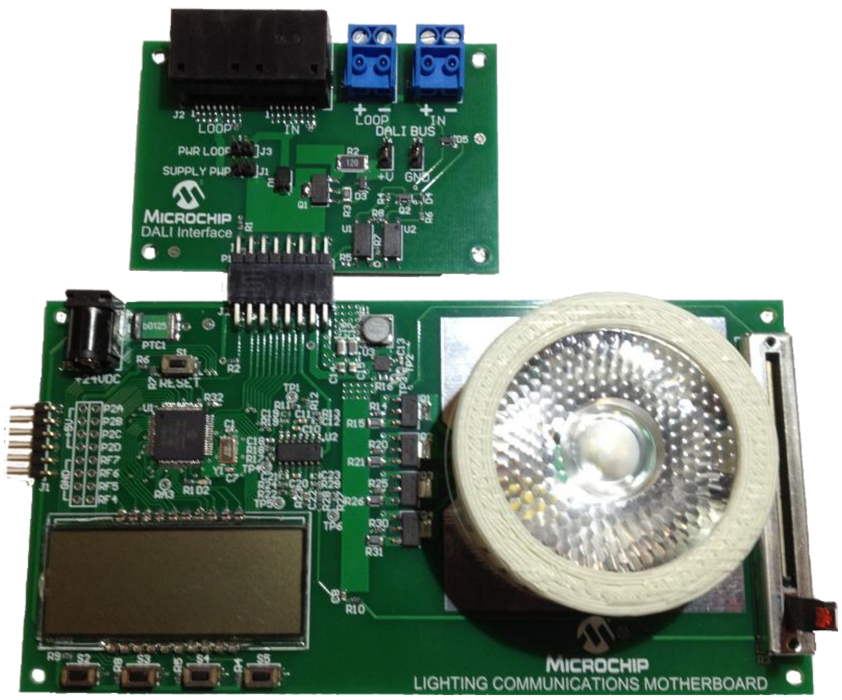
Lab 4:
DALI Control Gear

Lab 4 Objectives

- Set up Library for Control Gear
- Set up interrupt function
- Receive DALI data
- Automatic commissioning

Lab 4


LOOP
IN
DALI BUS
PWR LOOP
SUPPLY PWR
MICROCHIP
DALI Interface
RESET
MICROCHIP
LIGHTING COMMUNICATIONS MOTHERBOARD


LOOP
IN
DALI BUS
PWR LOOP
SUPPLY PWR
MICROCHIP
DALI Interface
RESET
MICROCHIP
LIGHTING COMMUNICATIONS MOTHERBOARD

Lab 4 Summary

- Easy to set up hardware
- Easy to add to Interrupt routine
- Easy to receive DALI data
- Easy to automatically commission

Agenda

- Lighting Protocols Overview
- DMX512
- Lab 1 - DMX512 Controller
- Lab 2 - DMX512 Receiver
- DALI
- Lab 3 - DALI Control Device
- Lab 4 - DALI Control Gear
- **Large Lighting Network**
- Lab 5 - Implementing a Larger Lighting Network
- Summary

Large Lighting Networks
Introduction

- **Why do we need large networks?**
  - To Save Money!!
    - Power & Maintenance cost money and for factories, cities (street lights), hotels, etc.; it can be a large expense.
  - Fast and accurate response to faults
  - Advanced control
  - Large area to cover
  - More lights than standard lighting buses allow
  - Remote operation and monitoring

Large Lighting Networks
Introduction

- **Places that need Large Networks**
  - City Street Lights
  - Large industrial buildings – factories & offices
  - Parking garages
  - Hotels and conference centers
  - Airports – in terminal and on runway
  - Theme parks
  - Architectural large scale illumination
  - Concerts and shows
  - Las Vegas!!!

Large Lighting Networks
Introduction

- **Common protocols used**
  - ZigBee
  - Ethernet
  - KNX
  - Cellular
  - Mixed
  - Proprietary

Large Lighting Networks
Issues

- Set up of individual addresses
- Wireless protocols have wireless issues (Range, Interference etc).
- Wired network has copper & install cost
- Complexity of routing data
- Complexity of control and monitoring
- Security

Agenda

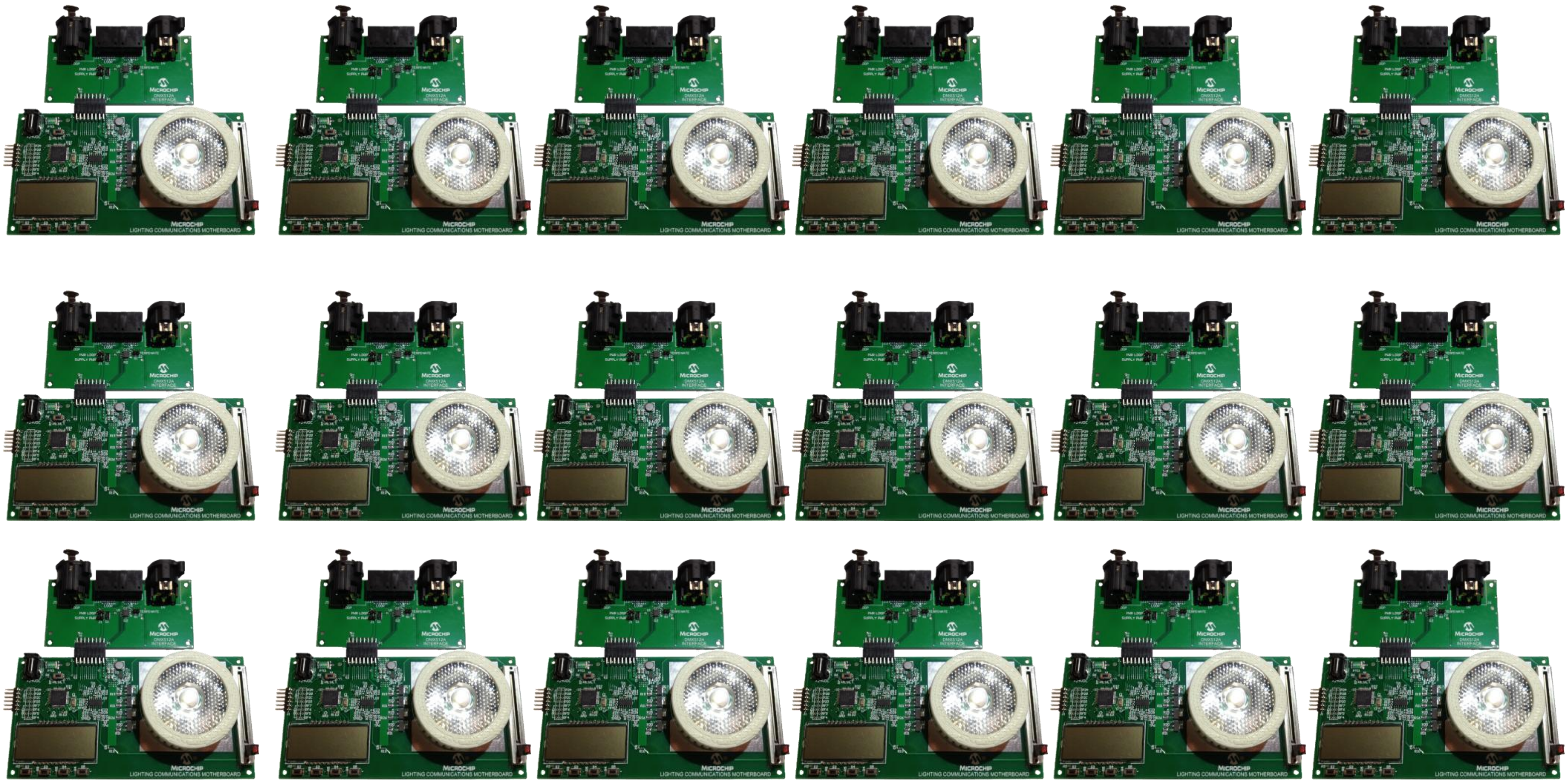
- Lighting Protocols Overview
- DMX512
- Lab 1 - DMX512 Controller
- Lab 2 - DMX512 Receiver
- DALI
- Lab 3 - DALI Control Device
- Lab 4 - DALI Control Gear
- Large Lighting Network
- **Lab 5 - Implementing a Larger Lighting Network**
- Summary

Lab 5:
Implementing a larger lighting network

Lab 5 Objectives

- Set up a larger network using DMX512
- Find issues with cabling
- Find issues with addressing
- Have a cool light show if it works

Lab 5



Lab 5 Summary

- Addressing Critical
- Cabling & Termination critical
- Good when it works
- Cool light show for the finale

Agenda

- Lighting Protocols Overview
- DMX512
- Lab 1 - DMX512 Controller
- Lab 2 - DMX512 Receiver
- DALI
- Lab 3 - DALI Control Device
- Lab 4 - DALI Control Gear
- Large Lighting Network
- Lab 5 - Implementing a Larger Lighting Network
- **Summary**

Summary

- **Today we covered:**
  - General Lighting Protocols
  - DMX512 and our Library
  - DALI and our Library
  - Overview of large lighting networks

Notice

We are Engineers, not Lawyers.

We are here to help you learn about our products, libraries, tools and to have some fun.

We understand that there may be potential patent concerns with different aspects of your projects. Please contact your own legal advisors for patent and legal advice.

Hardware Information

- **Lighting Communications Motherboard:**

  Part number DM160214

- **DMX512A Adapter:**

  Part Number AC160214-2

- **DALI Adapter:**

  Part number AC160214-1

Additional Resources
DMX 512

- **ANSI Standards**
  - http://webstore.ansi.org/
  - E1.11-2008 DMX512A Standard (~$40)
  - E1.20-2006 DMX512 Remote Device Management (~$40)
  - E1.27-1-2006 DMX512 Portable cabling Requirements (~$15)
- **App notes**
  - ANxxx – DMX512
  - ANxxx – Using the DMX512 Library
- **Websites**
  - http://www.dmx512-online.com/
  - http://en.wikipedia.org/wiki/DMX512
  - http://en.wikipedia.org/wiki/Digital_Addressable_Lighting_Interface

Additional Resources
DALI

- **IEC Standards**
  - http://www.iec.ch/
  - IEC 62386-101
  - IEC 62386-102
  - IEC 60929
- **App notes**
  - ANxxx – DALI Ballast
  - ANxxx – Using the DALI Library
- **Websites**
  - http://en.wikipedia.org/wiki/Digital_Addressable_Lighting_Interface
  - http://www.dali-ag.org/

Additional Resources
Misc

- **ArtNet**
  - *http://www.artisticlicence.com/WebSiteMaster/User Guides/**art-net**.pdf*
- **Ethernet**
  - http://www.microchip.com/ethernet
- **ZigBee**
  - http://www.zigbee.org
  - http://www.microchip.com/zigbee
- **MiWi**
  - http://www.microchip.com/miwi
- **Wi-Fi®**
  - http://www.microchip.com/wifi
- **KNX**
  - http://www.knx.org

LEGAL NOTICE

SOFTWARE:
You may use Microchip software exclusively with Microchip products. Further, use of Microchip software is subject to the copyright notices, disclaimers, and any license terms accompanying such software, whether set forth at the install of each program or posted in a header or text file.

Notwithstanding the above, certain components of software offered by Microchip and 3rd parties may be covered by “open source” software licenses – which include licenses that require that the distributor make the software available in source code format. To the extent required by such open source software licenses, the terms of such license will govern.

NOTICE & DISCLAIMER:
These materials and accompanying information (including, for example, any software, and references to 3rd party companies and 3rd party websites) are for informational purposes only and provided “AS IS.” Microchip assumes no responsibility for statements made by 3rd party companies, or materials or information that such 3rd parties may provide.

MICROCHIP DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, OR STATUTORY, INCLUDING ANY IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY DIRECT OR INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL, OR CONSEQUENTIAL LOSS, DAMAGE, COST, OR EXPENSE OF ANY KIND RELATED TO THESE MATERIALS OR ACCOMPANYING INFORMATION PROVIDED TO YOU BY MICROCHIP OR OTHER THIRD PARTIES, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBLITY OF SUCH DAMAGES OR THE DAMAGES ARE FORESEEABLE.

TRADEMARKS:
The Microchip name and logo, the Microchip logo, dsPIC, FlashFlex, KEELOQ, KEELOQ logo, MPLAB, PIC, PICmicro, PICSTART, PIC^{32} logo, rfPIC, SST, SST Logo, SuperFlash and UNI/O are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

FilterLab, Hampshire, HI-TECH C, Linear Active Thermistor, MTP, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Silicon Storage Technology is a registered trademark of Microchip Technology Inc. in other countries.

Analog-for-the-Digital Age, Application Maestro, BodyCom, chipKIT, chipKIT logo, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, HI-TIDE, In-Circuit Serial Programming, ICSP, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, mTouch, Omniscient Code Generation, PICC, PICC-18, PICDEM, PICDEM.net, PICkit, PICtail, REAL ICE, rfLAB, Select Mode, SQI, Serial Quad I/O, Total Endurance, TSHARC, UniWinDriver, WiperLock, ZENA and Z-Scale are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

GestIC and ULPP are registered trademarks of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

All other trademarks mentioned herein are property of their respective companies.