

Forward

This document describes OpenDMX Protocol, the extension of the USITT DMX512-A (from nowon DMX512A) protocol. The OpenDMX has been created to support extended features offered by the modern lighting systems, and not defined by the DMX512A protocol. This document is not mented to describe DMX512A protocol. For all information about DMX512A protocol, refer to “Entertainment Services and Technology Association, American National Standard E1.11 – 2004, Entertainment Technology, USITT DMX512-A, Asynchronous Serial Digital Data Transmission Standard for Controllino Lighting Equipment and Accessories” document (www.usitt.org).

1. Introduction

The DMX512A protocol is asynchronous, 250Kbit 8N2 communication based on RS485 electric standard. One single DMX512A communication frame is composed of :

- BREAK signal
- MARK after BREAK signal
- Start code byte
- Up to 512 DATA bytes

as shown in the following figure:

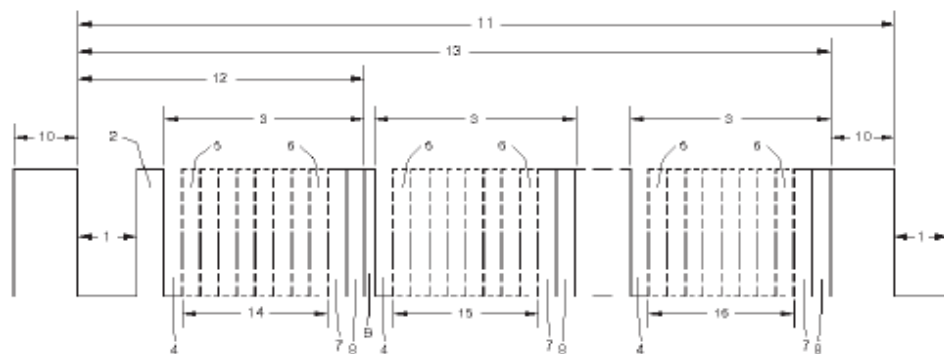


Figure Key

- 1 - "SPACE" for BREAK
- 2 - "MARK" After BREAK (MAB)
- 3 - Slot Time
- 4 - START Time
- 5 - LEAST SIGNIFICANT Data Bit
- 6 - MOST SIGNIFICANT Data Bit
- 7 - STOP Bit
- 8 - STOP Bit
- 9 - "MARK" Time Between Slots
- 10 - "MARK" Before BREAK (MBB)
- 11 - BREAK to BREAK Time
- 12 - RESET Sequence (BREAK, MAB, START Code)
- 13 - DMX512 Packet
- 14 - START CCODE (Slot 0 Data)
- 15 - SLOT 1 DATA
- 16 - SLOT nnn DATA (Maximum 512)

(source: ANSI E1.11 - 2004, Entertainment Technology - USITT DMX512-A - Asynchronous Serial Digital Data Transmisson, Standard for Controlling Lighting Equipment and Accessories CP/1998-1031r8.0)

The DMX512A protocol defines certain reserved START code bytes, shown in the following table:

Reserved Start codes	Description
0x00	Standard START code – Standard Operating Mode (SOM)
0x17	Text packet
0x4C	LUX OpenDMX start code
0x55	Text packet
0x90	Future expansion
0x91	Manufacturer ID field (2 bytes)
0x92 – 0xA9	Future expansion
0xAB – 0xCD	Future expansion
0xCF	System information
0xF0 – 0xF7	Prototyping codes

All OpenDMX frames will be transmitted using 0x4C (ASCII “L”) code.


1.1. Controller behaviour in absence of DMX signal

In absence of DMX transmission, every OpenDMX controller must behave as follows:

After POWER ON, if DMX transmission is absence for more then 3 seconds, controller should start with “demo” mode, fading from OFF to ON all possible colors that is capable to produce. The color order and fading duration depend on controller capabilities and are left free. This feature should be configurable via ODMX_CONFIG command.

If OpenDMX transmission is interrupted during normal operation, after more then 3 seconds, controller should turn ON at 50% of its output power, all outputs (simulating WHITE color). This feature also should be configurable via ODMX_CONFIG command.

If DMX signal will return, controller must immidiatley start to execute DMX commands.

	OpenDMX Protocol Specification	Document PHX 601-01	Rev. 1.0	Pag. 3/25
---	---------------------------------------	-------------------------------	--------------------	---------------------

2. OpenDMX protocol

2.1. OpenDMX transport layer

OpenDMX uses a unique format transport layer to transfer data between controlling and receiving unit:

<BREAK><OPENDMX_FRAME><ADDRESS16><BYTECOUNT16> [DATA0 . . . DATAn] <CHKSUM>

OPENDMX_FRAME = 0x4C

ADDRESS16 = Device address (0x0000...0xFFFFC). Address **0xFFFFE** will be used as **BROADCAST** (all devices must receive and process the command frame), two bytes (LSB transmitted first). Address **0xFFFFF** will be used as the Command Unit Address (**CU_ADDRESS**) for the reception frames. Every frame with the 0xFFFF address must be ignored by all receiving units. The **0xFFFFD** address (**ATTENTION**) is the special address, similar to broadcast. Any device that will receive frame with the 0xFFFFD address must process the frame and generate the answer (if needed).

BYTECOUNT16 = Number of DATA bytes, 16 bit (values 1...507), LSB byte transmitted first. OpenDMX permits data length up to **507 bytes**, so the total number of transmitted bytes does not increase 512.

DATA = DataBytes (at least one byte, OpenDXM Command)

CHKSUM = Checksum, XOR between all received bytes (included OPENDMX_FRAME and DATAn).

Using this transport layer, all OpenDMX commands will be transmitted.

Checksum example:

<BREAK><0x4C><0xFF><0xFE><0x01><0x00><0xF0><0xBC>

All 16 bit (or more) data bytes will be transmitted with LSB first.

In case of the data transfer that requires feedback frame (the answer from the receiving unit), the receiving unit must generate reception frame with the following contents:

<BREAK><OPENDMX_FRAME><0xFF><0xFF><BYTECOUNT16> [DATA0 . . . DATAn] <CHKSUM>

The answer frame must be generated and its transmission must start within 100 mS from the last byte of the transmission frame that generated it.

There will be no answer generated in case of request frame with a Broadcast address (in order to avoid data collision on the bus).

2.2. OpenDMX Command Review

OpenDMX data field is composed of at least ONE BYTE (OpenDMX Command). Command can be followed by OpenDMX data (for example, ODMX_CONFIG command) or can be stand alone (as ODMX_DIAG). Some commands will request the feedback generation (for example ODMX_STATUS).

Code	Command Name	Description
0x01	ODMX_CONFIG	Send device configuration
0x02	ODMX_ADDRESS	Set device address
0x10	ODMX_SETPWM	Set PWM outputs
0x11	ODMX_LOADPWM	Load new PWM for next activation
0x12	ODMX_ACTIVATE	Loaded PWM will be activated immediately
0x13	ODMX_PRFADE	Prepare outputs for a fading cycle
0x20	ODMX_FILESEND	Send file
0x21	ODMX_SWITCH	Send Switch status
0x22	ODMX_SRVANW	Service Answer
0x23	ODMX_SETSWITCH	Set Switch status
0x30	ODMX_STDBY	Toggle Stand by mode
0x40	ODMX_STATANW	Status answer
0xF0	ODMX_DIAG	Initialize Diagnostic cycle
0xF1	ODMX_STATUS	Request device status
0xF2	ODMX_GETSWITCH	Request Switch status
0xF3	ODMX_SRVREQ	Service Request
0xFE	ODMX_SIMPLEACK	Simple ACK without data field

This list of OpenDMX command is not complete. The OpenDMX workgroup will add in the future new commands. If you need to add new commands to the OpenDMX protocol, please contact OpenDMX workgroup on info@luxitalia.eu.

2.3. ODMX_CONFIG: Device Configuration

OpenDMX controller can operate in various modes. This command puts device in new operating mode.

The command has the following form:

```
<ODMX_CONFIG><CONFIGURATION_DATA>
```

Configuration data is the structure shown as follows:

```
typedef struct {
    unsigned char    RunMode;
    unsigned char    R;
    unsigned char    G;
    unsigned char    B;
    unsigned char    Y;
    unsigned char    PWM_Freq;
    unsigned char    Flags;
}CONFIGURATION;
```

RunMode

RunMode is the 8 bit filed that describes the operating mode of the device:

Mode	Description	WindowWidth
0	One/four channel 8 bit PWM controller	1
1	Two/Two channels 8 bit PWM controller	2
2	Three channels 8 bit PWM controller	3
3	Four channels 8 bit PWM controller	4
4	One channel White (RGBY) 8 bit controller	1
5	One channel 16 bit PWM controller	2
6	Two channels 16 bit PWM controller	4
7	Four channels 8 bit PWM + additional control bytes	8
8	Automatic mode 0 – All channels to max. value	4
9	Automatic mode 1 – All channels to RGBY value	4
10	Automatic mode 2 – Execute program from EEPROM in Single mode	----
11	Automatic mode 3 – Execute program from EEPROM in Master mode	----
12		----
13		----
14		----
15		----
255	Leave RUN mode unchanged	----

For more information about run modes, refer to Chapter 3, Run Modes.

RGBY fields

The four RGBY fields are used to set percentage of the RGBY channels in construction of the WHITE color. Those fields can have values from 0 to 100 (%). Value 255 means leave RGBY fields unchanged.

PWM_Freq

PWM_Freq is a register that configure output PWM frequency, according to the following table:

PWM 8 bit

PWM_Freq value	PWM Frequency	PWM Pulse Min 8 bit
0	93.6 KHz	41.7 nS
1	46.8 KHz	83.3 nS
2	23.4 KHz	167 nS
3	11.7 KHz	333 nS
4	5.85 KHz	667 nS
5	250 Hz	15 uS
6	150 Hz	52 uS
7	100 Hz	78 uS
8	30 Hz	128 uS

Note: Value 255 will leave PWM frequency unchanged.

PWM 16 bit

PWM_Freq value	PWM Frequency	PWM Pulse Min 16 bit
0	366 Hz	41.7 nS
1	183 Hz	83.3 nS
2	91 Hz	167 nS
3	46 Hz	333 nS
4	22 Hz	667 nS
5, 6, 7, 8	0,11 Hz	128 uS

Note: Value 255 will leave PWM frequency unchanged.

Flags register

This field is 8 bit flags register:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
---	---	---	---	---	---	SDL	SDM



- Bit 0 – SDM (Show DMX Missing): When set, this bit flag activate “Show DMX Missing” function. After Power On, if no DMX signal is found within 3 seconds from boot, controller will start with the demonstration program, turning OFF and ON with fading all cahnnels. Under “DMX Missing” is described any condition when no valid DMX frame has been received, no mather if the correct address has been received or not.
- Bit 1 – SDL (Show DMX Lost): When set, this bit flag activate “Show DMX Lost” function. If DMX signal has been received, and then lost or missing (but at least one transmission has been received), after 3 seconds from the DMX missing, controller will turn ON all PWM outputs on 50%. Under “DMX Lost” is described any condition when no valid DMX frame has been received, no mather if the correct address has been received or not.
- Bit 7 – Color flag bit. Driver that will drive single color lamp will have this bit set to 0. If a color lamp will be driven, this bit will be set to 1. This flag is used in automatic (demo) mode to define a type of demo program to execute.

Answer:

When received and processed, if ATTENTION address has been used, the unit must generate the answer to the DMX master. The answer will be simple ACK. The answer must be generated within 10 mS from the reception of the command

2.4. ODMX_ADDRESS

The ODMX_ADDRESS sets the position of the device within DMX network (device address) and sets WindowWidth.

The command has the following form:

```
<ODMX_ADDRESS><ADDRESS_DATA>
```

Configuration data is the structure shown as follows:

```
typedef struct {  
    unsigned int    TerminalAddress;  
}ADDRESS;
```

TerminalAddress

TerminalAddress is the 16 bit field that indicates device position within DMX network. This field can have any value from 0 to 65533. For the standard DMX protocol, valid values are from 0 to 511.

Answer:

When received and processed, if ATTENTION address has been used, the unit must generate the answer to the DMX master. The answer will be simple ACK. The answer must be generated within 10 mS from the reception of the command

2.5. ODMX_SETPWM

The ODMX_SETPWM is used to set PWM outputs for one or all nodes without need to send one 512 bytes standard DMX frame.

The command has the following form:

<ODMX_SETPWM><PWM_DATA>

PWM_DATA is structure with the following format:

```
typedef struct {
    unsigned int    Channel_1;
    unsigned int    Channel_2;
    unsigned int    Channel_3;
    unsigned int    Channel_4;
}PWM_DATA;
```

The structure presume that controller has 4 x 16 bit PWM outputs. Controllers that has 8 bit outputs will use only UPPER 8 bits of the 16 bit register value.

If controller has less then 4 channels, data will be ignored.

Controller in RUN_MODE_4 will use only Channel 1 member.

2.6. ODMX_FILESEND

This command is used to transfer a file (for example, user program). Since file can be longer then 500 bytes, the command includes multiply frame transfer. File can be long up to 32767 bytes.

The command has the following form:

<ODMX_FILESEND><SEND_HDR><SEND_DATA>

SEND_HDR is structure with the following format:

```
typedef struct {
    unsigned char    FileType;
    unsigned char    FrameFlags;
    unsigned int     Offset;
    unsigned int     DataLen;
}SEND_HDR;
```

FileType is any value from 0 to 255, and it indicate the “name” of the file. User program will have value “0”.

FrameFlags is 8 bit register with single bit flags:

Bit 0 – First frame. When set, this flag indicate that this frame is the first frame.

Bit 1 – Last frame. When set, this flag indicate that this frame is the last frame. No other frames will be received for this file.

Offset is the address inside the file where data will be written.

DataLen is the nuber of bytes inside SEND_DATA array.

SEND_DATA array is the array of file data.

NOTE: You must allow 2 mS delay (EEPROM writing) for every transmitted file byte before to transmit next FILESEND command. In meantime you can refresh DMX anyway.

2.7. ODMX_STDBY

This command is used to put device in stand by mode, or to wake up device from stand by mode.

In stand by mode, device should turn off all outputs and enter low power mode. During stand by mode, no activity on DMX bus will be present. Master device can, anyway, send cyclically a “refresh” frame of the stand by command (for example, every 2-3 seconds) in order to keep other devices informed that DMX network is operating.

The command has the following form:

```
<ODMX_STDBY><STDBY_OFF/STDBY_ON>
```

```
STDBY_OFF = 0x55
```

```
STDBY_ON = 0xAA
```

Any OpenDMX command except SWITCH commands received during stand by mode will wake up device.

2.8. ODMX_DIAG

The ODMX_DIAG is the diagnostic command. It is used to initialize DIAGNOSTIC cycle. After receiving this command, DMX device will memorize all output values, execute diagnostic cycle, save the diagnostic results and restore outputs. No feedback answer will be generated at the end of the diagnostic cycle. Diagnostic results will be saved and returned back through ODMX_STATUS command.

During diagnostic cycle DMX device will ignore DMX protocol for the whole duration of the diagnostic cycle. The maximum diagnostic duration is fixed to 1 second.

2.9. ODMX_STATUS

The ODMX_STATUS is the command used to discover the status of every single DMX node. After receiving this command, device will generate status feedback, and send it back to the DMX controller. The answer contains three types of information: real-time status, last diagnostic result and configuration information.

Real time status is the information about hardware of the device. Diagnostic status is the result of the last executed diagnostic. Configuration information is used to inform the control unit about internal configuration.

For devices with the unknown address, the 0xFFFD address can be used in ODMX_STATUS command to discover device address.

The feedback frame has the following format:

```
<BREAK><OPENDMX_FRAME><CU_ADDRESS16><BYTECOUNT16><ODMX_STANW><STATUS_STRUCT>  
<CHKSUM>
```

```
OPENDMX_FRAME = 0x4C
```

```
CU_ADDRESS16 = 0xFFFF
```

```
BYTECOUNT16 = sizeof(STATUS_STRUCT)
```

```
typedef struct {  
    unsigned char    bPowerSupply;  
    unsigned char    bStatus;  
    char             cFwVersion[4];  
    unsigned int     iDiagData[4];  
};
```

```

unsigned int      iDeviceAddress;
unsigned char     RunMode;
unsigned char     R;
unsigned char     G;
unsigned char     B;
unsigned char     Y;
unsigned char     PWM_Freq;
}STATUS_STRUCT;

```

bPowerSupply is 8 bit register that represents level of the input power supply. It can have value from 0 to 255. Every decimal value represents 200 mV resolution. For example, bPowerSupply = 87 means that input power supply measured by internal ADC has value of 17.4V. Maximum power supply level is 50.8V.

NOTE: power supply level must be measured with all PWM channels at maximum output. Data in this register are valid only when PSVALID flag in bStatus register is set.

bStatus is 8 bit register:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CHN_NUM		PSVALID	X	X	X	X	EE_OK

CHN_NUM – indicates number of physical output channels available on board:

Bit 7	Bit 6	Channel number
0	0	1
0	1	2
1	0	3
1	1	4

Even if device is configured in 4 x 16 bit PWM output channel, but it can be configured as 4 x 8 bit PWM output, it must respond as 4 channel output device.

PSVALID – When set (1), indicate that bPowerSupply register contains valid data

EE_OK field – When set (1), indicates EEPROM good.

cFwVersion is a non terminated, 4 bytes ASCII string that indicate firmware version, in format

Byte 0	Byte 1	Byte 2	Byte 3
1	.	2	3

This value will be decided as “Version 1.23”.

iDiagData is 4 x 16 bit array. Each 16 bit register is the result of diagnostic on one output channel:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
bOutputCurrent							

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
DVALID	X	X	X	X	X	LSHORT	LOPEN



DVALID – When set (1), indicate that iDiagData register contains valid information. When this flag is zero, other iDataDiag information should be ignored.

LOPEN – When set (1), indicate that output current is NULL - Load is opened

LSHORT – When set (1), indicate that output is in short circuit

bOutputCurrent – 8 bit field that indicates maximum channel output current, measured when channel is on maximum PWM. The resolution is 0.04 Amp (40 mA), and the maximum current is 10.16 Amp.

iDeviceAddress is the DMX address of the device (16 bit).

RunMode is the run mode in which device is operating (see chapter “3. Run Modes”)

R,G,B,Y fields are the fields for White color creation (see chapter “2.3 ODMX_CONFIG: Device Configuration”)

PWM_Freq – see “2.3 ODMX_CONFIG: Device Configuration”

2.10. ODMX_GETSWITCH

This command is used to read remote color / dimmer / switch controller. Remote controller must be able to give following information:

- System state (OFF/ON)
- Four channel color regulation (R,G,B,Y)
- Program selection

Obviously, controller that is not supporting color selection, must generate color field anyway.

The command has the following form:

```
<ODMX_GETSWITCH>
```

Switch must have address range from 0xFFF0 to 0xFFF7 (up to seven switches), and it must answer within 10 mS from the request.

The answer will be transmitted using ODMX_SWITCH frame.

2.11. ODMX_SWITCH

The ODMX_SWITCH command is used to transfer remote switch color / dimmer / switch controller status. Answer has following form:

```
<ODMX_SWITCH><SWITCHSTAT>
```

```
typedef struct {  
    unsigned char    State;  
    unsigned char    R;  
    unsigned char    G;  
    unsigned char    B;  
    unsigned char    Y;  
    unsigned char    Dimmer;  
    unsigned char    Program;  
}SWITCHSTAT;
```

State can be 0 (OFF) or 1 (ON). If MSB of the State register is set, indicate that new data are changed from the last data transmission.

R,G,B,Y are 0...255 levels of 4 channel color

Program is selected program value (0 means no program selected, 1-127 indicate program ID)

If **Program** value has MSB set, indicate that selected program is dynamic program, while when reset indicates static program.

2.12. ODMX_SETSWITCH

The ODMX_SETSWITCH command is used to set remote switch state.

```
<ODMX_SWITCH><SWITCHSTAT>
typedef struct {
    unsigned char    State;
    unsigned char    R;
    unsigned char    G;
    unsigned char    B;
    unsigned char    Y;
    unsigned char    Dimmer;
    unsigned char    Program;
}SWITCHSTAT;
```

2.13. ODMX_SRVREQ

This command is used by the Master device on DMX network to request other remote devices if they have something to say (for example, during programming or so on).

The command must be addressed to the specific address, that is **0xFFF8**. This address is reserved to the remote device that can be inserted to the DMX network from time to time.

Once sent **Service Request** command, a device that sent it must wait for 10 mS eventual answer. The answer will arrive under ODMX_SRVANW command.

2.14. ODMX_SRVANW

The command ODMX_SRVANW is used by the remote device when it has some service activity to signal to the master device. This command must be issued only during ODMX_SRVREQ 10 mS period.

The format is following:

```
<ODMX_SRVANW><SERVICE_ID>
```

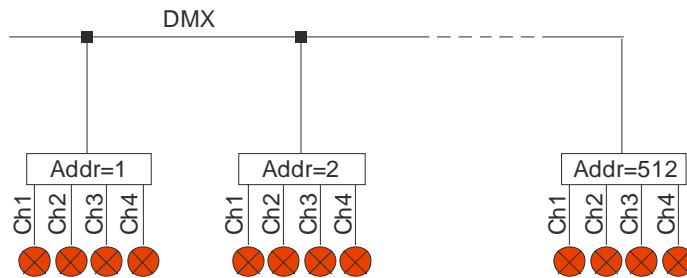
SERVICE_ID	Description
0x00	Nothing to do
0x01	Enter File transfer mode. The master device must halt with any bus activity, while the remote device that issued the answer must start with the file transfer not before 1mS and within 10 mS from the transmission of the service answer.

3. Run Modes

OpenDMX controller could support more than one run mode. For this reason Configuration command is capable to set different run modes. Devices that support only one run mode should ignore RunMode filed. In this chapter will be discussed all run modes.

3.1. Run Mode 0

Run mode 0 is the simplest run mode. It presumes that DMX node (device) has only one output channel, so only one DMX data byte will be of interest for device.



If device has more than one output channel, all channels will receive the same PWM settings given for the first channel.

Example:

DMX SOM Command:

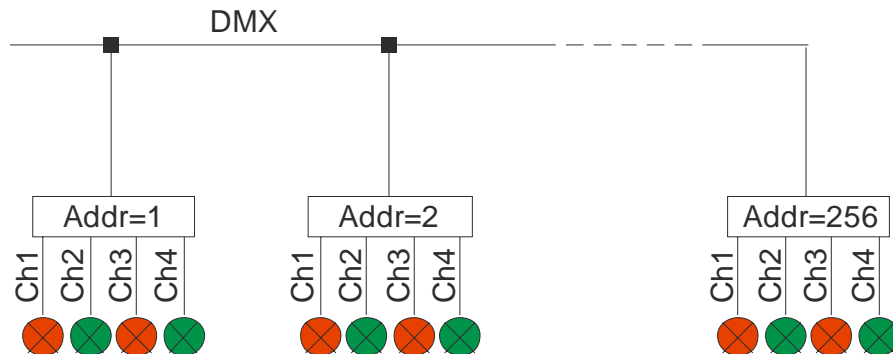
`<BREAK><0x00><0x20><0x40>.....<0xF0>`

Channel settings after SOM reception:

Device 1				Device 2				Device 512			
Ch1	Ch2	Ch3	Ch4	Ch1	Ch2	Ch3	Ch4	Ch1	Ch2	Ch3	Ch4
0x20	0x20	0x20	0x20	0x40	0x40	0x40	0x40	0xF0	0xF0	0xF0	0xF0

3.2. Run Mode 1

Run mode 1 presumes that DMX node (device) has two output channels. Two bytes will be acquired from the SOM DMS frame.



If device has four output channels, channels 3 and 4 will assume the same values of channels 1 and 2 (channel 3 from channel 1, and channel 4 from channel 2).

Example:

DMX SOM Command:

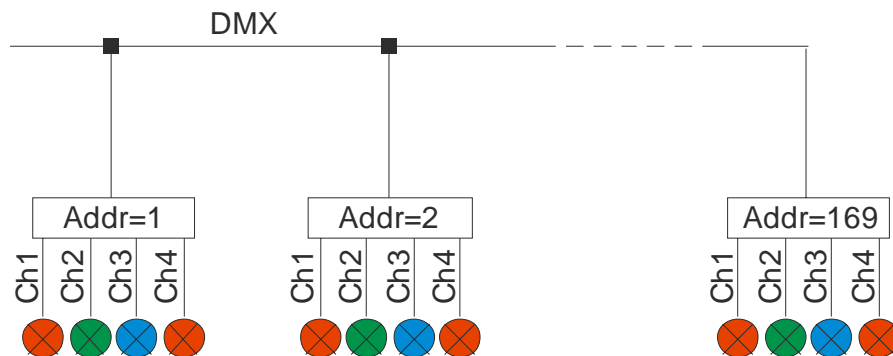
<BREAK><0x00><0x20><0x30><0x40><0x50> <0xE0><0xF0>

Channel settings after SOM reception:

Device 1				Device 2				Device 256			
Ch1	Ch2	Ch3	Ch4	Ch1	Ch2	Ch3	Ch4	Ch1	Ch2	Ch3	Ch4
0x20	0x30	0x20	0x30	0x40	0x50	0x40	0x50	0xE0	0xF0	0xE0	0xF0

3.3. Run Mode 2

Run mode 2 presumes that DMX node (device) has three output channels. Three bytes will be acquired from the SOM DMX frame. In this running mode the full RGB can be reached.



If device has four output channels, channel 4 will assume the same value of channel.

Example:

DMX SOM Command:

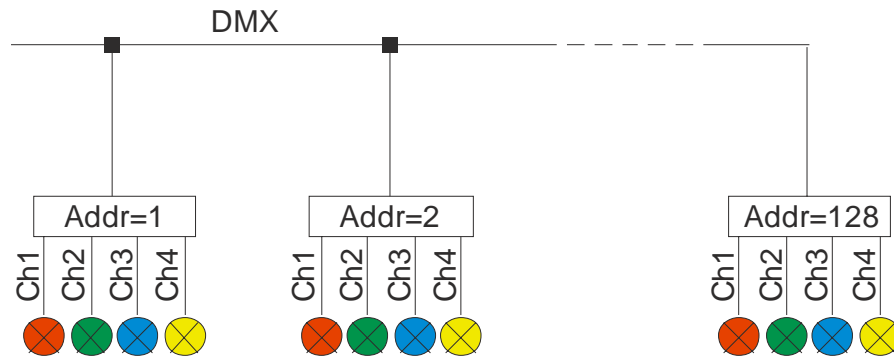
<BREAK><0x00><0x21><0x22><0x23><0x31><0x32><0x33> <0xF1><0xF2><0xF3>

Channel settings after SOM reception:

Device 1				Device 2				Device 169			
Ch1	Ch2	Ch3	Ch4	Ch1	Ch2	Ch3	Ch4	Ch1	Ch2	Ch3	Ch4
0x21	0x22	0x23	0x21	0x31	0x32	0x33	0x31	0xF1	0xF0	0xF3	0xF1

3.4. Run Mode 3

Run mode 3 is also called “Full OpenDMX” mode, since it is capable to drive 4 PWM, 8 bit channels. Four bytes will be acquired from the SOM DMX frame.



Example:

DMX SOM Command:

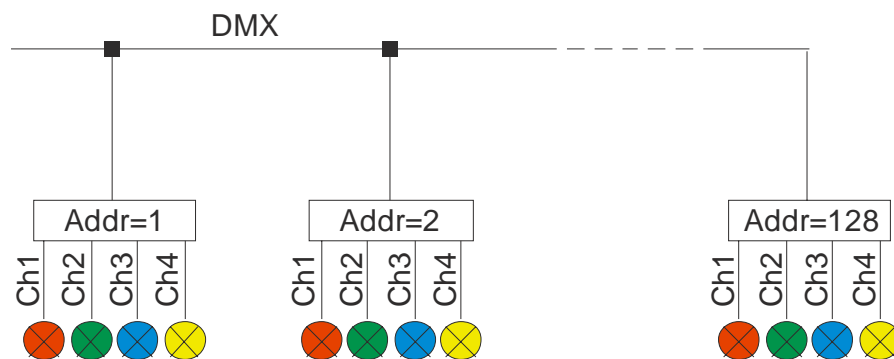
```
<BREAK><0x00><0x21><0x22><0x23><0x24><0x31><0x32><0x33><0x34> . . . . . <0xF1><0xF2><0xF3><0xF4>
```

Channel settings after SOM reception:

Device 1				Device 2				Device 128			
Ch1	Ch2	Ch3	Ch4	Ch1	Ch2	Ch3	Ch4	Ch1	Ch2	Ch3	Ch4
0x21	0x22	0x23	0x24	0x31	0x32	0x33	0x34	0xF1	0xF0	0xF3	0xF4

3.5. Run Mode 4

Run mode 4 is used in those applications where the light source is RGBY, while the final requested result from the light source is only a White color. In this way, with only one SOM data byte it is possible to regulate White color from four RGBY PWM, 8 bit channels. Before to use this operating mode, the RGBY settings must be done, in order to define exact ratio of single RGBY components in the White color creation.



Example:

First, the RGBY ratio must be set using ODMX_CONFIG command. Suppose that RGBY ratio is:

$C_r = 40 \text{ dec}$, $C_g = 100 \text{ dec}$, $C_b = 80 \text{ dec}$, $C_y = 20 \text{ dec}$

means: 40% of RED, 100% of GREEN, 80% of BLUE and 20% of YELLOW

DMX SOM Command:

<BREAK><0x00><0x20><0x30>.....<0xF0>

means: Channel 1 is at 32 dec (Sr = 32, 0 is minimum, 255 is maximum), Channel 2 is at Sr=48 dec, Channel 512 is at Sr=240 dec of maximum intensity

Calculating percentage for the channel 1, that must be dimmed to 0x20 (32 decimal), and knowing that 100% of the output power is 255 decimal, the output on all four channels R G B Y will be:

$$R = (Cr * Sr) / 100 = (40 * 32) / 100 = 12 \text{ dec} = 0x0C$$

$$G = (100 * 32) / 100 = 32 \text{ dec} = 0x20$$

$$B = (80 * 32) / 100 = 25 \text{ dec} = 0x19$$

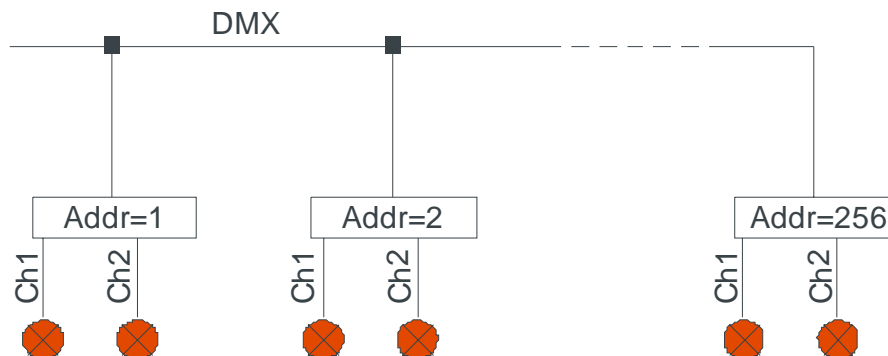
$$Y = (20 * 32) / 100 = 6 \text{ dec} = 0x06$$

Channel settings after SOM reception:

Device 1				Device 2				Device 128			
Ch1	Ch2	Ch3	Ch4	Ch1	Ch2	Ch3	Ch4	Ch1	Ch2	Ch3	Ch4
0x0C	0x20	0x19	0x06	0x13	0x30	0x26	0x09	0x60	0xF0	0xC0	0x30

3.6. Run Mode 5

Run mode 5 offers 16 bit resolution. For this purpose, two bytes will be acquired from the SOM DMX frame. The first byte will be acquired as LSB.



If device has two 16 bit PWM channels, the second PWM channel will assume the same value as the first channel.

Example:

DMX SOM Command:

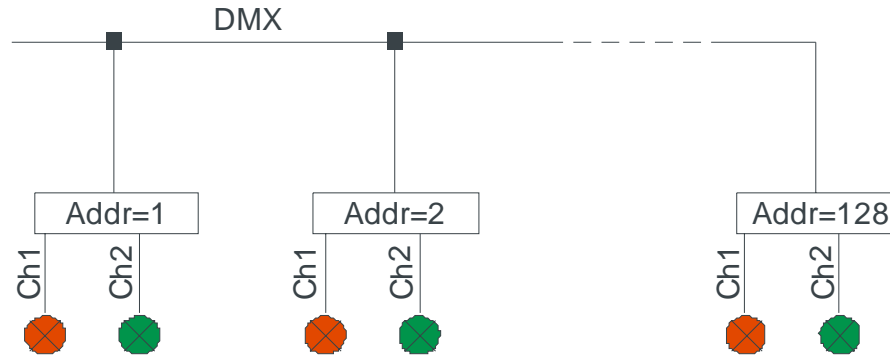
<BREAK><0x00><0x21><0x22><0x23><0x24>.....<0xF1><0xF2>

Channel settings after SOM reception:

Device 1		Device 2		Device 256	
Ch1	Ch3	Ch1	Ch3	Ch1	Ch3
0x2221	0x2221	0x2423	0x2423	0xF2F1	0xF2F1

3.7. Run Mode 6

Run mode 6 offers two 16 bit channels. For this purpose, four bytes will be acquired from the SOM DMX frame. The first byte will be acquired as LSB.



Example:

DMX SOM Command:

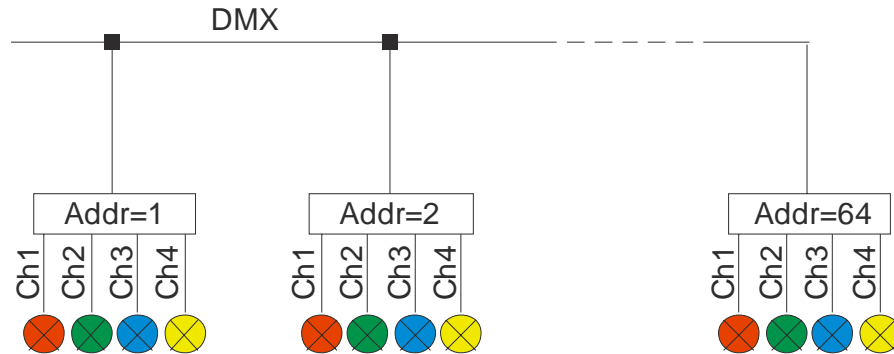
```
<BREAK><0x00><0x21><0x22><0x23><0x24><0x25><0x26><0x27><0x28> . . . . . <0xF1><0xF2><0xF3><0xF4>
```

Channel settings after SOM reception:

Device 1		Device 2		Device 128	
Ch1	Ch3	Ch1	Ch3	Ch1	Ch3
0x2221	0x2423	0x2625	0x2827	0xF2F1	0xF4F3

3.8. Run Mode 7

Run mode 7 is advanced DMX 8 bit mode. It has 4 channel 8 bit PWM regulation, including 4 additional control bytes.



For each DMX node, 8 bytes will be acquired from the SOM DMX frame:

```
typedef struct {
    unsigned char    R;
    unsigned char    G;
    unsigned char    B;
    unsigned char    Y;
    unsigned char    Sync;
    unsigned char    Strobe;
    unsigned char    Delay;
    unsigned char    Reserved;
}DMX_NODE;
```

3.9. Run Mode 8

In this mode driver operates in autonomous mode. All 4 channels will be set to max. value (100% PWM). If DMX will be connected, driver will behave as in Run Mode 3.

3.10. Run Mode 9

In this mode driver operates in autonomous mode. All 4 channels will be set to value stored in R,G,B,Y fields of the configuration. If DMX will be connected, driver will behave as in Run Mode 3.

3.11. Run Mode 10

In this mode driver operates in autonomous mode, executing program from the EEPROM. The program will be stored using ODMX_FILESAVE command. File has following structure:

Field Description	Address				
File Header	0x00	RecNumLO	RecNumHI	ChnQtaLO	ChnQtaHI
Record 1	0x04	One record contents, 10 bytes			
				
Record N	4 + (10 x N)	One record contents, 10 bytes			

RecNumLO/HI is 16 bit value total number of records in the file.

ChnQtaLO/HI is the number of active channels of the file. Since this mode is intended to control only four driver channels, this value is fixed to 4.

Record is the structure with the contents as follows:

```
typedef struct {
    unsigned char    RecType;
    unsigned char    Parameter;
    unsigned int     Fade;
    unsigned int     Wait;
    unsigned char    Outputs[4];
}RECORD;
```

RecType is the field that explains the type of the record. There are following record types:

Record Type	RecType Value	Description
RT_SHOW	0	Record that will be used to set the outputs
RT_GOTO	1	When arrive to this record, driver will jump to the record described in Parameter field. Program execution will continue from the newly record.
RT_CALL	2	When arrive to this record, driver will jump to the record described in Parameter field. Program execution will continue from the newly record.
RT_RETURN	3	When arrive to this record, driver will return to the record of the last CALL. Up to 32 calls/returns can be computed.
RT_REPEAT	4	When arrive to this record, driver will continue to execute all program records until it will find RT_LOOP record. Number of loops is defined with Parameter field.
RT_LOOP	5	Closes RT_REPEAT loop. Up to 32 nested repeat loops can be defined.

RT_SHOW record description

This record type defines driver outputs. For this record, **Parameter** field is not used.

Fade field is the time for output metamorphosis. During this time, outputs will change from previous PWM value to the newly PWM value defined with **Outputs** fields. This time is expressed in milliseconds, and can be 0. Setting this time to 0, driver will change immediately outputs to the new value.

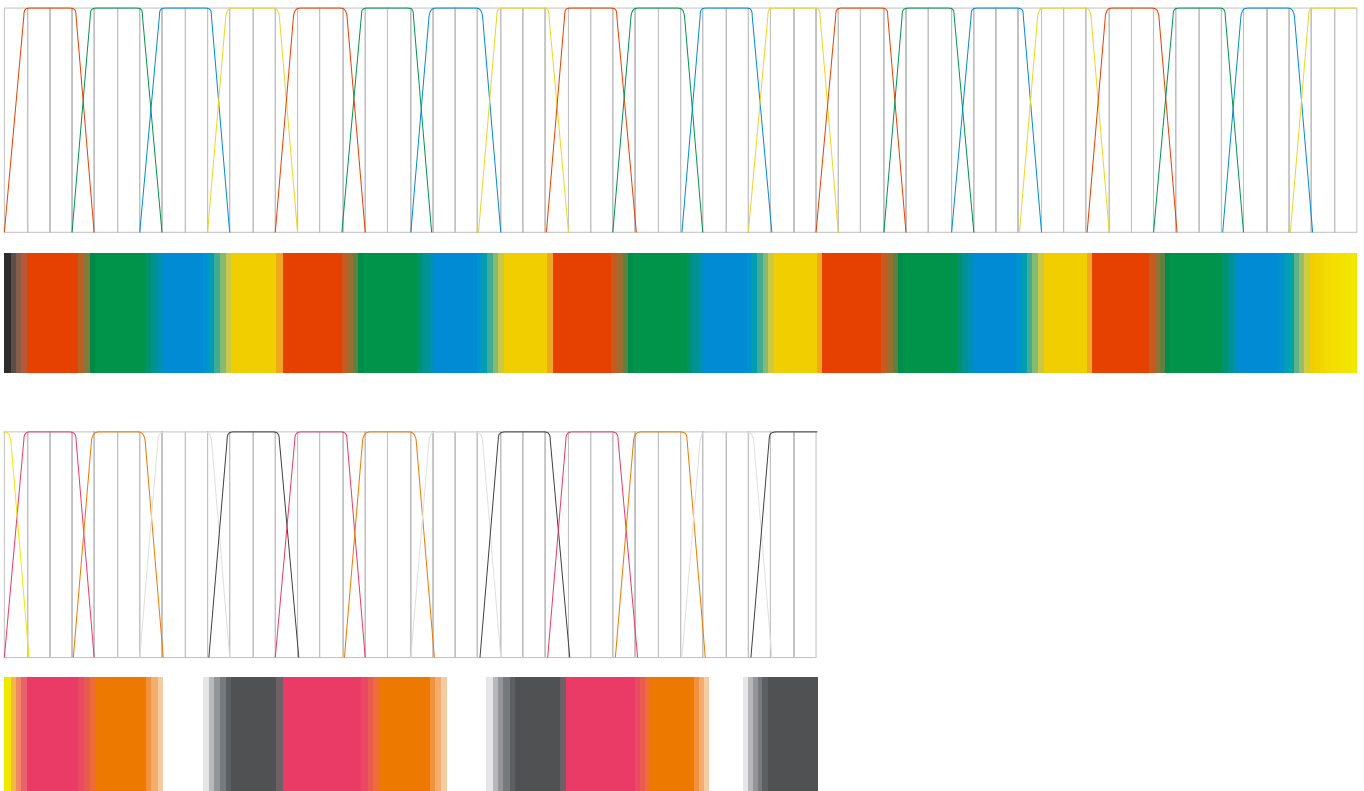
Wait field is the time for the record duration, once when outputs have been settled. During wait time, no changes to outputs will be done. For values below 32768, this time is expressed in milliseconds, and can be 0. Setting this time to 0, driver will step immediately to the next record. For values above 32767, value is expressed in seconds, by resetting MSB of the value to 0 (value 32769 will be interpreted as $32769 - 32768 = 1$ second).

Outputs[4] are 4 channels values, from 0 to 255.

Program File Example

Here will be shown one program example. It will be explained step-by-step, then described using “C” language, and at the end shown as a physical code to be saved in the EEPROM of the driver.

This example program will show 4 steps with single colours (Red, Green, Blue, Yellow) repeated 5 times, then it will show for 3 times combination of Magenta-Orange-White-Dark Gray, and then it will start from the beginning. Graph shows one full cycle:



Note: Timebase is 1 second (every grid rectangle takes 1 sec).

The same program can be described using “C” language:


```
// Note: Two functions, "SetOutput" and "Wait" used below are not shown.
// They have following format:
//     void SetOutput(int R, int G, int B, int Y, int FadeTime)
//     void Wait(int WaitTime);
//
// *****
// Main body of the user program
// *****
void user_program(void){
    int i;

    while(1){ // Program main loop (forever)

        for(i=0; i<5; i++){ // First loop, 5 times
            SetOutput(255, 0, 0, 0, 1000); // Set RED channel
            Wait(2000); // Wait 2 seconds (2000 ms)
            SetOutput(0, 255, 0, 0, 1000); // Set GREEN channel
            Wait(2000); // Wait 2 seconds (2000 ms)
            SetOutput(0, 0, 255, 0, 1000); // Set BLUE channel
            Wait(2000); // Wait 2 seconds (2000 ms)
            SetOutput(0, 0, 0, 255, 1000); // Set YELLOW channel
            Wait(2000); // Wait 2 seconds (2000 ms)
        } // End of the first loop

        for(i=0; i<3; i++){ // Second loop, 3 times
            sub1(); // Execute subroutine 1
            sub2(); // Execute subroutine 2
        } // End of the second loop

    } // End of the main loop
} // End of the user program
// *****
// Subroutine 1
// *****
void sub1(void){
    SetOutput(255,0,255,0,1000); // Set MAGENTA
    Wait(2000); // Wait 2 seconds (2000 ms)
    SetOutput(255,0,0,255,1000); // Set ORANGE
    Wait(2000); // Wait 2 seconds (2000 ms)
}
// *****
// Subroutine 1
// *****
void sub1(void){
    SetOutput(255,0,255,0,1000); // Set MAGENTA
    Wait(2000); // Wait 2 seconds (2000 ms)
    SetOutput(255,0,0,255,1000); // Set ORANGE
    Wait(2000); // Wait 2 seconds (2000 ms)
}
}
```


	OpenDMX Protocol Specification	Document PHX 601-01	Rev. 1.0	Pag. 25/25
---	---------------------------------------	-------------------------------	--------------------	----------------------

3.12. Run Mode 11

Similar to RunMode 10, driver in this mode operate in autonomous mode, executing program from the EEPROM, but it also controls (becoming DMX master) up to 12 external channels (3 devices with 4 channels). The program will be stored using ODMX_FILESAVE command. File has the same structure as the file in RunMode 10, except that number of channels can be from 4 to 16. First 4 channels are those connected to the driver, while other 12 channels will be transmitted using standard DMX frame.