IS3715: I2C DMX512 Controller

Full Universe, 512 Channels

Applications

- Custom Lighting Controllers
- Automated Light Controllers
- Digital Art Installations
- Architectural & Building Lighting
- Stage & Entertainment Lighting
- Animatronics
- Water fountains
- OEM / Device Manufacturers
- Museum Lighting

Main Advantages

- Reduces engineering time and costs
- Reduces product time-to-market
- Makes the DMX protocol transparent to both the microcontroller and the engineer
- Provides a low-cost solution
- Fewer ISRs, lower microcontroller CPU load
- Reduces microcontroller memory usage
- Saves microcontroller dedicated pins with I2C
- Minimizes impact on microcontroller peripherals (no need for dedicated timers, UARTs, etc.)
- Compact, easy-to-solder SO8N package
- I2C speeds: 100 kHz, 400 kHz, and 1 MHz

DMX Characteristics

- DMX512-A Protocol Compliant
- Sends All 512 Channels
- Synchronization Output Pin
- Update rate: 44 Hz

General Description

The IS3715 is an I2C DMX Controller chip. You write values to its internal memory map via I2C and it continuously outputs them as DMX data at 44 Hz.

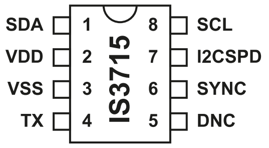
The chip features 512 8-bit RAM registers to store DMX channel values, which can be written to like a typical I2C EEPROM. It continuously outputs all memory as DMX data, so you only need to write via I2C when values change, keeping your microcontroller relaxed.

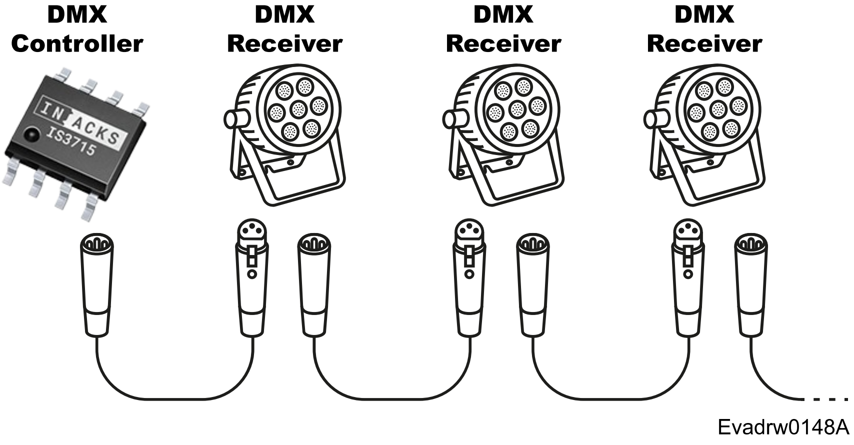
The I2C-Serial interface operates as a slave and supports 100 kHz, 400 kHz, and 1 MHz communication speeds.

The IS3715 frees your microcontroller from the DMX protocol timing constraints, memory requirements, and ISR load associated with generating DMX data, reducing the need for timers, flash, and RAM. It also eliminates the need for dedicated pins by operating over a shared I2C.

The device operates at 3.3 V, with 5 V-tolerant I2C, enabling compatibility with 3.3 V or 5 V microcontrollers. It is offered in Industrial (–40 °C to +85 °C) and Extended (–40 °C to +125 °C) temperature ranges.

Part Number	Package	Op. Temperature
IS3715-S8-I	SO8N	-40ºC to +85ºC
IS3715-S8-E	SO8N	-40ºC to +125ºC





1 Electrical Specification

Absolute Maximum Ratings

Parameter			Min	Max	Unit
Input Voltage	VDD Pin		-0.3	4	V
	SCL, SDA, TX, and SYNC Pins		-0.3	5.5	
	I2CSPD Pin		-0.3	4	
Current Sourced/Sunk by any I/O or Control Pin				±20	mA
Temperature	Operating Temperature	IS3715-S8-I	-40	+85	ºC
		IS3715-S8-E	-40	+125	
	Storage Temperature		-65	+150	
Electrostatic Discharge (TA = 25ºC)	Human-body model (HBM), Class 1C		-2000	+1500	V
	Charged-device model (CDM), Class C2a		-500	+500	

Exceeding the specifications outlined in the Absolute Maximum Ratings could potentially lead to irreversible harm to the device. It's important to note that these ratings solely indicate stress limits and don't guarantee the device's functionality under such conditions, or any others not specified in the Recommended Operating Conditions. Prolonged exposure to conditions at or beyond the absolute maximum ratings might compromise the reliability of the device.

Recommended Operation Conditions

Parameter	Symbol	Min	Nom	Max	Unit
Supply Voltage	V_{DD}	2.0	3.3	3.6	V
Input Voltage at SCL, SDA. TX and SYNC Pins	$V_{I/O-IN}$	-0.3	3.3	5.5	
Input Voltage at I2CSPD Pin	$V_{I2CSPD-IN}$	-0.3	1.8, 3.3	3.6	
Source/Sink Current at SCL, SDA, TX and SYNC Pins	$I_{I/O-SS}$	-	-	±6	mA

Electrical Characteristics

Parameter		Symbol	Min	Nom	Max	Unit
Current Consumption (T_A= 25ºC)		I_{OP}	-	3.40	3.90	mA
Input Voltage	Logical High-Level	V_{IH}	0.7xV_{DD}	-	-	V
	Logical Low-Level	V_{IL}	-	-	0.3xV_{DD}	

Electrical Specifications Revision A

2 Detailed Description

2.1 IS3715 Description

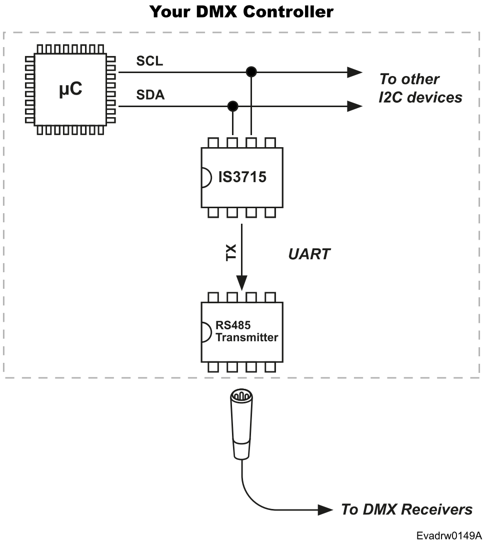
The IS3715 is a DMX512 protocol-specialized integrated circuit that generates DMX data from the contents of its memory map, which are updated via I2C from your microcontroller.

The DMX data can control any DMX-compatible device, usually including:

- Spotlights and moving heads
- LED strips, panels, and bars
- Stage lights
- Floodlights, wall washers, and PAR cans
- Lasers
- Fog, haze, smoke machines
- Water fountains
- Building facades, bridges, monuments
- Relays, and motorized devices, etc.

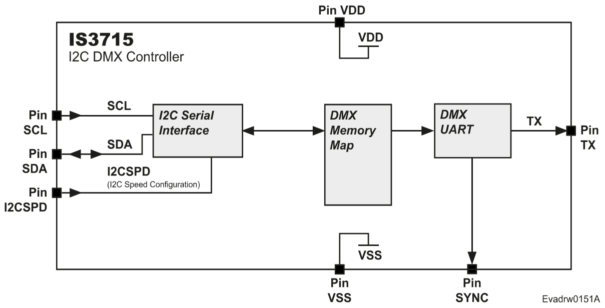
The DMX512 standard states that this protocol should never be used to control pyrotechnics or stage robotics, as it has no intrinsic error checking. Therefore, this chip should not be used for those applications either.

The chip connects to your microcontroller via I2C, where it acts as an I2C slave at the address 21 (0x15). To set a DMX channel, the microcontroller writes the corresponding memory address. For example, writing to register address `1` sets the value of DMX Channel 1, writing to register address `512` sets the value of DMX Channel 512, and so on for the full range of channels from `1` to `512`.



To set the I2C speed, connect the I2CSPD pin to GND for 100 kHz, to 1.65 V for 400 kHz (using a simple resistor voltage divider: 1.65 = 3.3 V ÷ 2), or to 3.3 V for 1 MHz. The SDA and SCL pins operate at 3.3 V logic levels and are 5 V tolerant.

The IS3715 continuously generates DMX data from its memory map without interruption. This has the advantage that if a cable picks up noise or a connector makes a poor contact, a glitch may appear on the lights, but it will be instantly cleared thanks to the IS3715 keeps updating the data continuosly. The update frequency is 44 Hz.



The DMX data is output on the TX pin at TTL voltage levels, while the DMX network requires RS485 voltage levels. Therefore, you must connect an RS485 driver or transceiver, such as the THVD1500, to this pin to convert TTL to RS485. You can find more details about the design in the Hardware Examples section.

The IS3715 features an an output synchronization pin which can be used to synchronize multiple DMX universes. It goes high for 100 µs at the beginning of the DMX Break signal. The use of this pin is optional and is not part of the DMX512 standard.

The IS3715 operates at 3.3 V. Bypass its VDD pin to GND with a 100nF ceramic capacitor.

2.2 Usage

To use the chip, your microcontroller must write DMX values to the IS3715 via I2C in to control DMX fixtures such as lights. For proper operation, first ensure that both hardware and firmware are correctly configured by following the validation steps. Once validated, you can begin controlling DMX from your firmware.

Hardware Validation

To ensure proper operation via I²C, verify the following hardware conditions:

1. The VDD pin is supplied with 3.3 V.
2. The SCL and SDA pins have pull-ups to 3.3 V or 5 V.
3. For an I²C speed of 100 kHz, the I2CSPD pin must be connected to VSS.
4. The TX pin is connected to an RS-485 driver or transceiver.
5. For a non-isolated controller design, ensure that your microcontroller, the IS3715, the transceiver, and pin 1 of the DMX connector (signal common) share the same ground.

If these conditions are met, you can safely begin using the chip through the I2C Serial interface.

Refer to chapter Hardware Examples for more information about hardware design.

Firmware Validation

Before setting values to the DMX channels, it is good practice to first validate that your microcontroller can properly communicate via I2C.

The simplest way to do this is to scan the I2C-Serial interface and confirm that the IS3715 is detected.

Once detected, you can be confident that the I2C-Serial interface is working correctly, and that any remaining issues will come from your firmware code.

At this point, you can implement your I2C write routine. Remember that the IS3715 uses 16-bit register addressing with 8-bit data size. To verify correct addressing, read the `CHIP_ID` register at address `513`. This register is read-only and always returns the fixed value `21 (0x15)`. If you obtain this value, communication is validated and you can safely begin sending DMX data via I²C to control the DMX devices.

1. Scan the I2C-Serial interface to confirm detection of the IS3715.
2. Read the `CHIP_ID` register (address `513`) and verify that it returns the fixed value `21`.

Refer to application note *ISAN0001-How to scan I2C Serial Interface* for instructions on scanning the I2C-Serial interface.

Refer to chapter Firmware Examples chapter for example codes.

2.3 I2C Timing vs DMX Timing

The IS3715 always updates all 512 DMX channels on its TX pin every 22.9278 ms, corresponding to 43.61 Hz. This rate is constant and does not change under any circumstance. Even if no new data is received via I2C, the TX pin continues refreshing at 43.61 Hz. Continuous refreshing helps clear glitches or artifacts that may appear on DMX receivers due to poor cable connections, electrical noise, or other issues affecting the RS-485 bus.

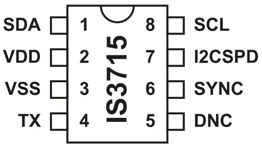
The time required to write DMX data via I2C depends on the number of channels being written, the I²C speed, and the microcontroller's processing time to configure and send the data.

Writing via I2C all the 512 DMX channels at 1 MHz takes ~4.635 ms, while the TX pin outputs a full DMX frame in 22.9278 ms. That allows the microcontroller to update the entire DMX universe at truly 43.61 Hz.

Writing via I2C all the 512 DMX channels at 400 kHz takes ~11.587 ms, compared to 22.9278 ms for the TX pin to output a full DMX frame. In this case, writing all DMX channels via I2C takes slightly more than than one TX frame.

Similar happens when writing all the 512 DMX channels via I2C at 100 kHz, which takes ~46.35 ms, while outputting a whole DMX frame via TX pin takes 22.9278 ms. In this case the IS3715 will need 3 frames to fully update all 512 DMX channels.

3 Pin Description



Pin	Name	Type	Description
1	SDA	Open Drain 5V Tolerant	I2C Data pin. Open drain, it requires pull-up.
2	VDD	Supply	3.3V power supply pin. Bypass this pin to GND with a 100nF ceramic capacitor.
3	VSS	Ground	Ground reference pin.
4	TX	Digital Output Push-Pull	DMX UART Transmit Pin in TTL voltage levels. Attention: Use an RS485 transmitter or transceiver to adapt the TTL voltage levels to DMX voltage levels.
5	DNC	Do Not Connect	Leave this pin unconnected.
6	SYNC	Digital Output Push-Pull	DMX Synchronization Pin.
7	I2CSPD	Analog Input 0 to 3.3V	I2C-Serial Interface Speed Selection pin. • For 100kHz pull to GND. • For 400kHz make a voltage divider of VDD/2 (1.65V). • For 1MHz pull to VDD (3.3V).
8	SCL	Open Drain 5V Tolerant	I2C Clock pin. Open drain, it requires pull-up.

3.1 TX Pin

DMX UART Transmit Pin.

This pin transmits DMX512 data to the RS485 transmitter or transceiver. It operates at 3.3V TTL levels.

Use an RS485 transmitter or transceiver, such as the THVD1400DR, to convert TTL voltage levels to DMX (RS485) differential signals. Refer to the Hardware Examples section for more details.

Important: Connecting this pin directly to DMX bus without the transmitter or transceiver will permanently damage the device.

3.2 SYNC Pin

DMX Synchronization Pin.

This pin can be used to synchronize multiple DMX universes. It goes high for 100 µs at the beginning of the Break signal.

The use of this pin is optional and is not part of the DMX512 standard.

3.3 SCL and SDA Pins

I2C-Compatible Bus Interface Pins.

SCL (Serial Clock Line): This pin is used to synchronize data transfer between the IS3715 device and the microcontroller or other CPU.

SDA (Serial Data Line): This bidirectional pin is used for both sending and receiving data between the IS3715 and the microcontroller or other CPU.

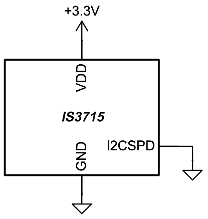
Both pins are open-drain and must be pulled up to 3.3 V or 5 V. The pull-up resistor value should be chosen based on the bus speed and capacitance. Typical values are 4.7 kΩ for Standard Mode (100 kbps) and 2 kΩ for Fast Mode (400 kbps) at both 3.3 V and 5 V.

3.4 I2CSPD Pin

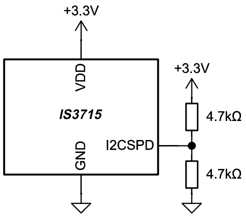
I2C-Serial Interface Speed Selection Pin.

This pin configures the IS3715 internal I2C-Serial Interface timings and filters to properly work with the selected bus speed.

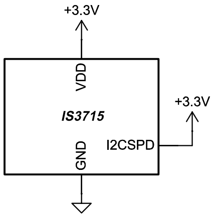
For a **100 kHz** setting, set the I2CSPD pin to VSS.



For a **400 kHz** setting, set the I2CSPD to 1.65 V (VDD/2) using a balanced voltage divider. This can be achieved by placing two 4.7 kΩ resistors from the I2CSPD pin: one to VDD and the other to VSS.



For a **1000 MHz** setting, set the I2CSPD pin to 3.3 V.



Important Remarks:

Voltages above 4 V on this can permanently damage the device.

A mismatch between the configured I2C speed and the actual operating I2C speed (e.g., setting I2CSPD to GND for 100 kHz but operating at 1 MHz) can lead to an inconsistent state where some I2C messages are processed while others are not.

Ensure a proper match between the actual operating speed and the configured speed at the I2CSPD pin: If your bus works at 100 kHz, ensure the I2CSPD pin is tied to VSS. If it works at 400 kHz ensure the pin is at 1.65 V. If it works at 1 MHz, ensure the pin is at 3.3 V.

4 Memory Map Description

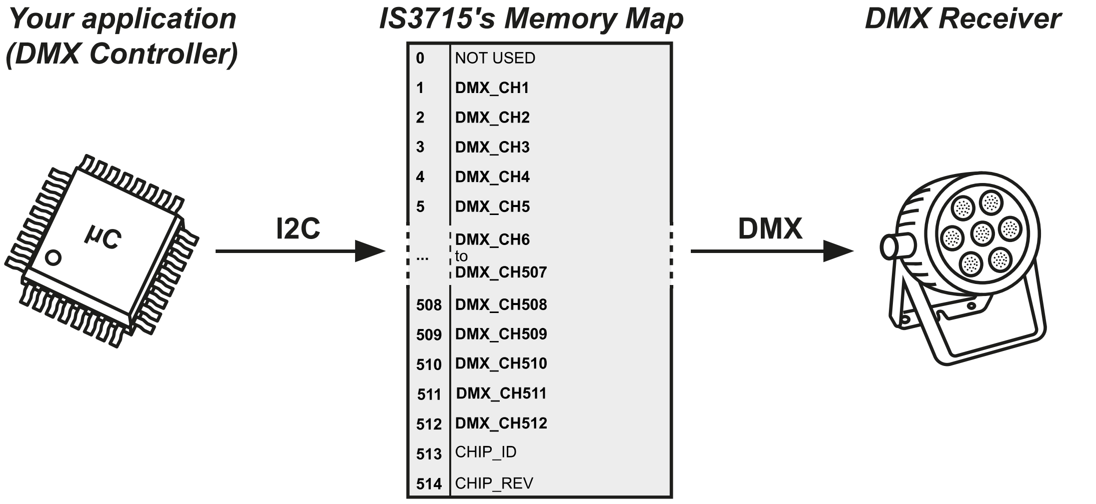
The IS3715 is internally organized as a single page containing 515 registers of 8 bits each, with addresses ranging from 0 to 514. Therefore, accessing any register requires a 2-byte addressing.

The memory type is RAM, which means you can read of write any register without page-block limitations.

There are two types of registers: the DMX channel registers (`DMX_CHx`) and the chip information registers.

The DMX channel registers are mapped in the memory map so that each register number corresponds its DMX channel number, making it easy for engineers to understand and address the memory. For example, DMX Channel 15 data is stored into the register 15.

The two chip information registers contain the `CHIP_ID`, which is fixed throughout the product's lifetime, and the `CHIP_REV`, which changes if the chip undergoes any revision. The `CHIP_ID` can be used for chip detection in the I2C-Serial Interface during firmware development



0	NOT USED
1	DMX_CH1
2	DMX_CH2
3	DMX_CH3
4	DMX_CH4
5	DMX_CH5
...	DMX_CH6 to DMX_CH507
508	DMX_CH508
509	DMX_CH509
510	DMX_CH510
511	DMX_CH511
512	DMX_CH512
513	CHIP_ID
514	CHIP_REV

Evadrw0150A

4.1 DMX_Chx Registers

The DMX_CHx registers contain the values that will be sent to the corresponding DMX channels. Each register number matches the DMX channel number. For example, writing to DMX_CH43 register will update the data of DMX Channel 43.

There are 512 DMX_CHx registers (DMX_CH1 to DMX_CH512), stored in volatile RAM. If the chip loses power, these registers reset to 0 on power-up until new data is written via I2C. You can access the registers individually or in blocks of any size. You can even update all DMX registers in a single I2C Multiple Byte Write operation.

These registers are R/W registers.

Name:	DMX_CHx
Description:	DMX Channel Values
Address Range:	1 to 512 (0x001 to 0x200)
Memory Type:	Volatile RAM
Allowed Values:	0 to 255 (0x00 to 0xFF)
Reset Values:	0 (0x00)

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

4.2 CHIP_ID Registers

The CHIP_ID register contains the chip identifier, which is a fixed value of 21 (0x15). This value is used for production tracking. It is stored in ROM and will not change throughout the product's life-cycle.

Since this register value is constant, reading it during firmware development can help verify that I2C communications are working and that the chip's memory can be properly read.

This register is read-only..

Name:	CHIP_ID
Description:	DMX Channel Values
Address:	513 (0x201)
Memory Type:	ROM
Value:	21 (0x15)

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

4.3 CHIP_REV Registers

The `CHIP_REV` register indicates the chip revision.

This value is intended for production and product tracking. It is stored in ROM and may change throughout the product's life-cycle.

This register is read-only.

Name:	`CHIP_REV`
Description:	Chip Revision Number
Address:	`514` (`0x202`)
Memory Type:	ROM
Value:	(Depends on the revision)

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

5 I2C Description

The IS3715 operates as a Slave in the I2C-Serial Interface. It supports Standard Mode (100 kHz), Fast Mode (400 kHz), and Fast Mode Plus (1 MHz). The I2C-Master device, typically a microcontroller or a microprocessor, initiates and manages all read operations to the Slave.

The IS3715 is represented on the bus by the I2C device address: 21 (0x15).

Pull-up resistors are required on the SCL and SDA lines for proper operation. The resistor values depend on the bus capacitance and operating speed. Typical values are 4.7 kΩ for Standard Mode (100 kHz) and 2 kΩ for Fast Mode and Fast Mode Plus (400 kHz and 1 MHz).

The IS3715's I2C pins high state can be either 3.3 V or 5 V. A logical '0' is transmitted by pulling the line low, while a logical '1' is transmitted by releasing the line, allowing it to be pulled high by the pull-up resistor. The Master controls the Serial Clock (SCL) line, which generates the synchronous clock used by the Serial Data (SDA) line to transmit data.

A Start or Stop condition occurs when the SDA line changes during the High period of the SCL line. Data on the SDA line must be 8 bits long and is transmitted Most Significant Bit First and Most Significant Byte First. After the 8 data bits, the receiver must respond with either an acknowledge (ACK) or a no-acknowledge (NACK) bit during the ninth clock cycle, which is generated by the Master. To keep the bus in an idle state, both the SCL and SDA lines must be released to the High state.

The memory map consists of 515 registers, each 8 bits wide. Addressing a register requires a 2-byte pointer (DMX Pointer Register).

The operability of the Read and Write commands of the IS3715 is very similar to an EEPROM memory. Thinking of the IS3715 as an EEPROM memory is a good analogy to quickly understand how to communicate with the device.

5.1 Highlights

I2C Device Address: 21 (0x15)

I2C Memory Map Addressing Size: 16-bit (2x 8-bit)

I2C Memory Map Register Size: 8-bit

Compatible I2C Speeds:
- Standard Mode (100 kHz), recommended SCL and SDA pull-up value: 4.7 kΩ
- Fast Mode (400 kHz), recommended SCL and SDA pull-up value: 2 kΩ
- Fast Mode (10 MHz), recommended SCL and SDA pull-up value: 2 kΩ

Supported Operations:
- Single-Byte Write
- Multiple-Byte Write (up to 515 registers)
- Single-Byte Read
- Multiple-Byte Read (up to 515 registers)

Overreading and Overwriting the memory:
- If a write operation starts at a valid memory address (`0` to `514`) and continues past the last valid address, it will roll over to address 0.
- Starting a write operation to an invalid memory address (greater than `514`) will result in a NACK and data will be discarded.
- If a read operation starts at a valid memory address (`0` to `514`) and continues past the last valid address, it will roll over to address 0.
- Starting a read operation at an invalid memory address (greater than `514`) will return a value of 0xFF.

5.2 Single Byte Write

Writing a single byte is an action performed by the microcontroller (I2C-Master) to write data to any register within the IS3715 memory (I2C-Slave), regardless of the last read or written position. To perform this action, the microcontroller must load the register address intended to be written into the IS3715's Pointer Register. Once the address is set, the Microcontroller can send the data to be written.

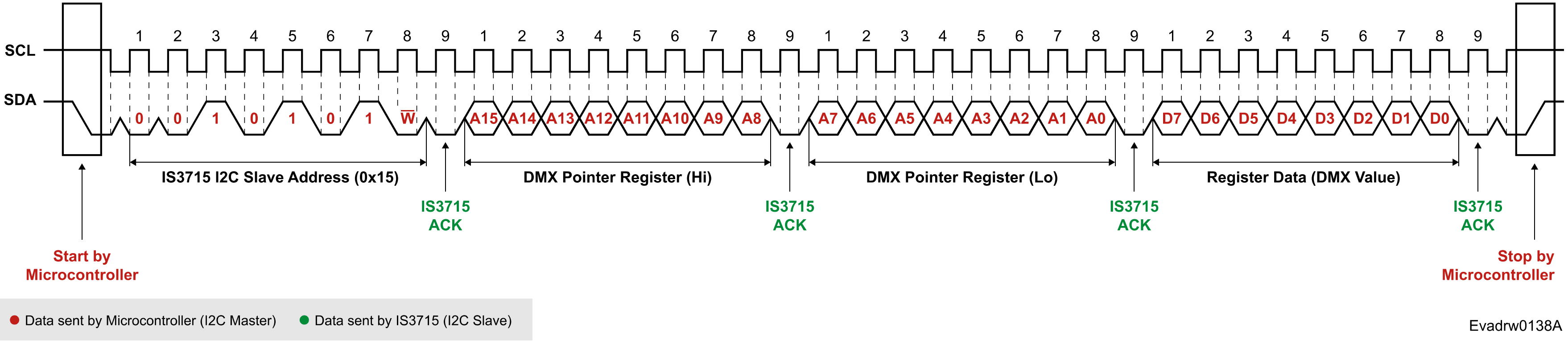
To initiate the Single Byte Write operation, the following steps must be performed from the beginning: The microcontroller begins by pulling down the SDA line while the SCL line is high, creating a Start Condition. It then sends the IS3715 I2C device address 21 (0x15) with the R/W bit set to `0` (indicating a write operation). Note that the IS3715's I2C address is fixed and does not change, allowing it to be uniquely identified among other devices on the I2C-Serial Interface.

Upon receiving the device address, the IS3715 acknowledges it. Subsequently, the microcontroller sends the two bytes of the register address it intends to write: the most significant byte first, followed by the least significant byte, each acknowledged by the IS3715.

The microcontroller then sends the byte to be written, which the IS3715 acknowledges. Finally, the microcontroller issues a Stop Condition by raising the SDA line while the SCL line is high.

Invalid Memory Addressing

The valid memory range of the IS3715 for a write operation goes from addresses `0` to `514`. If a Write Operation is performed with a Pointer Register higher than `514`, the IS3715 will answer with a NACK.


SCL
SDA
1 2 3 4 5 6 7 8 9
0 0 1 0 1 0 1 W̄
A15 A14 A13 A12 A11 A10 A9 A8
A7 A6 A5 A4 A3 A2 A1 A0
D7 D6 D5 D4 D3 D2 D1 D0
IS3715 I2C Slave Address (0x15)
DMX Pointer Register (Hi)
DMX Pointer Register (Lo)
Register Data (DMX Value)
IS3715 ACK
IS3715 ACK
IS3715 ACK
IS3715 ACK
Start by Microcontroller
Stop by Microcontroller
● Data sent by Microcontroller (I2C Master)
● Data sent by IS3715 (I2C Slave)
Evadrw0138A

5.3 Multiple Byte Write

The Multiple Byte Write operation functions similarly to the Single Byte Write, but allows writing a block of up to 515 registers in a single operation—that is, the entire memory in one go.

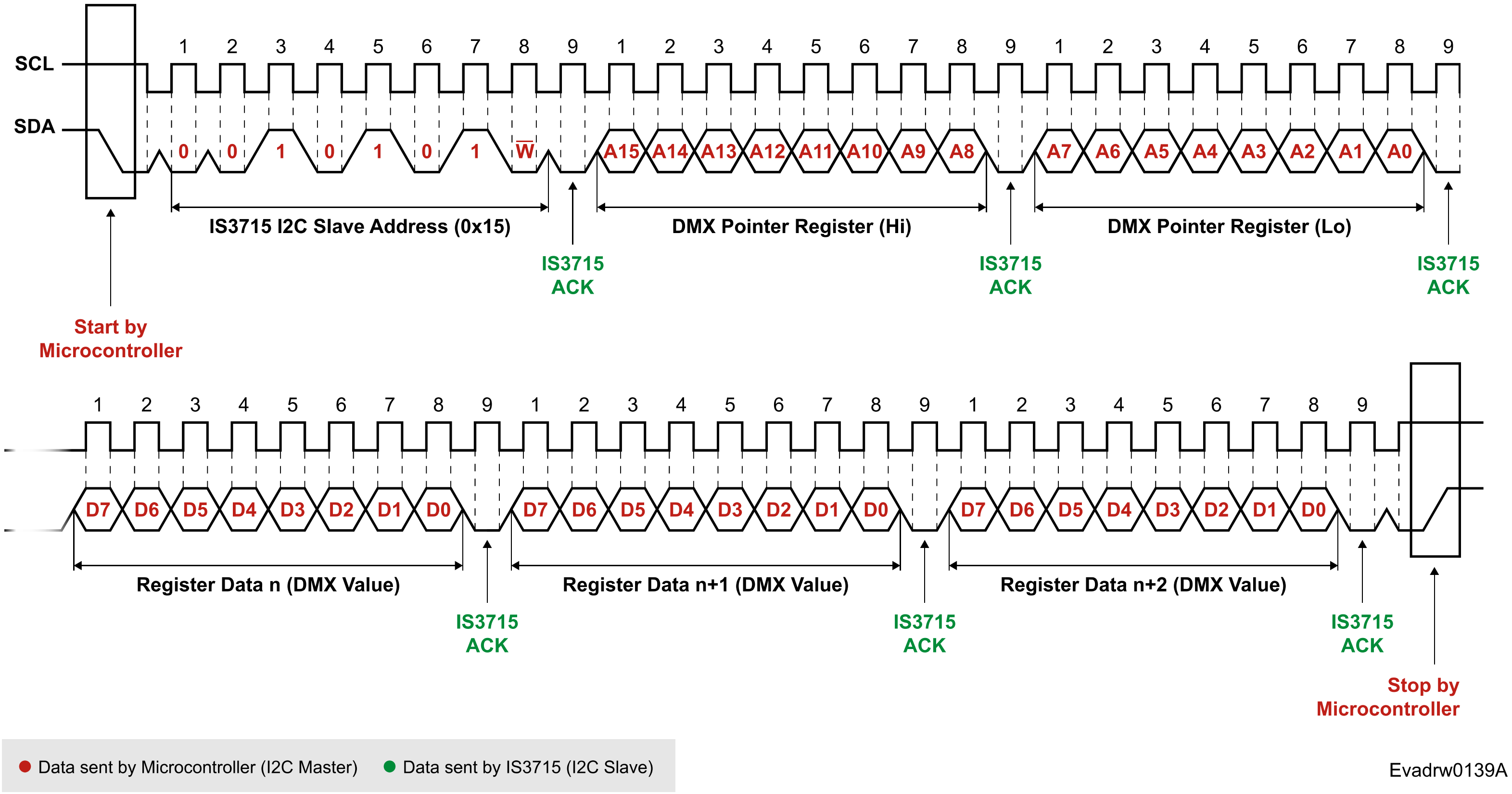
To perform a Multiple Byte Write operation, follow the same procedure as for a Single Byte Write until the first data byte is written. After writing the first byte, instead of generating a Stop Condition, the microcontroller should continue writing data bytes. To conclude the write operation, after sending the last data byte, the microcontroller should generate a Stop Condition.

Invalid Memory Addressing

The valid memory range of the IS3715 for a write operation goes from addresses `0` to `514`.

If a Multiple Byte Write Operation is performed with a Pointer Register within the valid memory range (`0` to `514`) but exceeds the last memory register (`514`), a rollover to register `0` will occur.

If a Multiple Byte Write Operation is performed with a Pointer Register outside the valid memory range (greater than `514`), the IS3715 will respond with a NACK upon receiving the first data byte.



5.4 Single Byte Read

Reading a single byte is an action performed by the microcontroller (I2C-Master) to access any register within the IS3715 memory (I2C-Slave), regardless of the last read position. To perform this action, the microcontroller must load the DMX channel into the IS3715's DMX Pointer Register. Once the address is set, the microcontroller can retrieve the DMX data from the specified register.

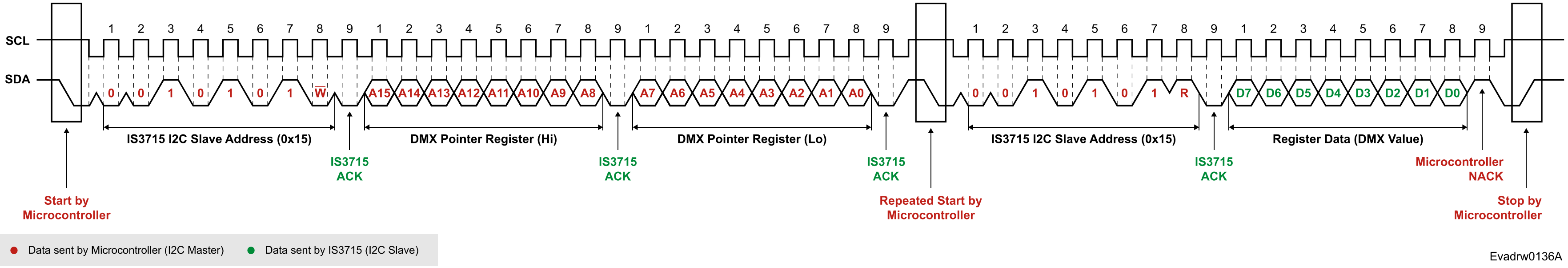
To initiate the Single Byte Read operation, the following steps must be performed from the beginning: The microcontroller starts by pulling SDA low while SCL is high to generate a Start Condition. It then sends the IS3715 I2C device slave address 21 (0x15) with the R/W bit set to `0` (write). Upon receiving the device address, the IS3715 acknowledges it. Subsequently, the microcontroller sends the two bytes of the DMX Pointer Register address: the most significant byte first, followed by the less significant byte, each acknowledged by the IS3715. This sets the address of the next DMX channel to be read in the DMX Pointer Register.

Next, the content of the DMX Pointer Register needs to be read.

The microcontroller generates a Repeated Start Condition, followed by the IS3715 I2C device address 21 (0x15) with the R/W bit set to `1` (read), instructing the IS3715 to retrieve data. The IS3715 acknowledges and responds with the DMX data, which the microcontroller does not acknowledge (NACK). Finally, the microcontroller issues a Stop Condition by raising the SDA line while the SCL is high.

Invalid Memory Addressing

The valid memory range of the IS3715 goes from addresses `0` to `514`. If a Read Operation is performed with a Pointer Register higher than `514`, the read result will be 0xFF.



5.5 Multiple Byte Read

Multiple Byte Read operation functions similarly to Single Byte Read but can read a block of up to 515 registers in a single operation, corresponding to the full memory map.

To perform a Multiple Byte Read operation, follow the same procedure as for a Single Byte Read until the first byte is received. After receiving the first byte, instead of generating a Not Acknowledge (NACK), the microcontroller should continue acknowledging (ACK) each received data byte from the IS3715 for as many bytes as it intends to read. To conclude the read operation, after reading the last data byte, the microcontroller should generate a Not Acknowledge (NACK) and a Stop Condition.

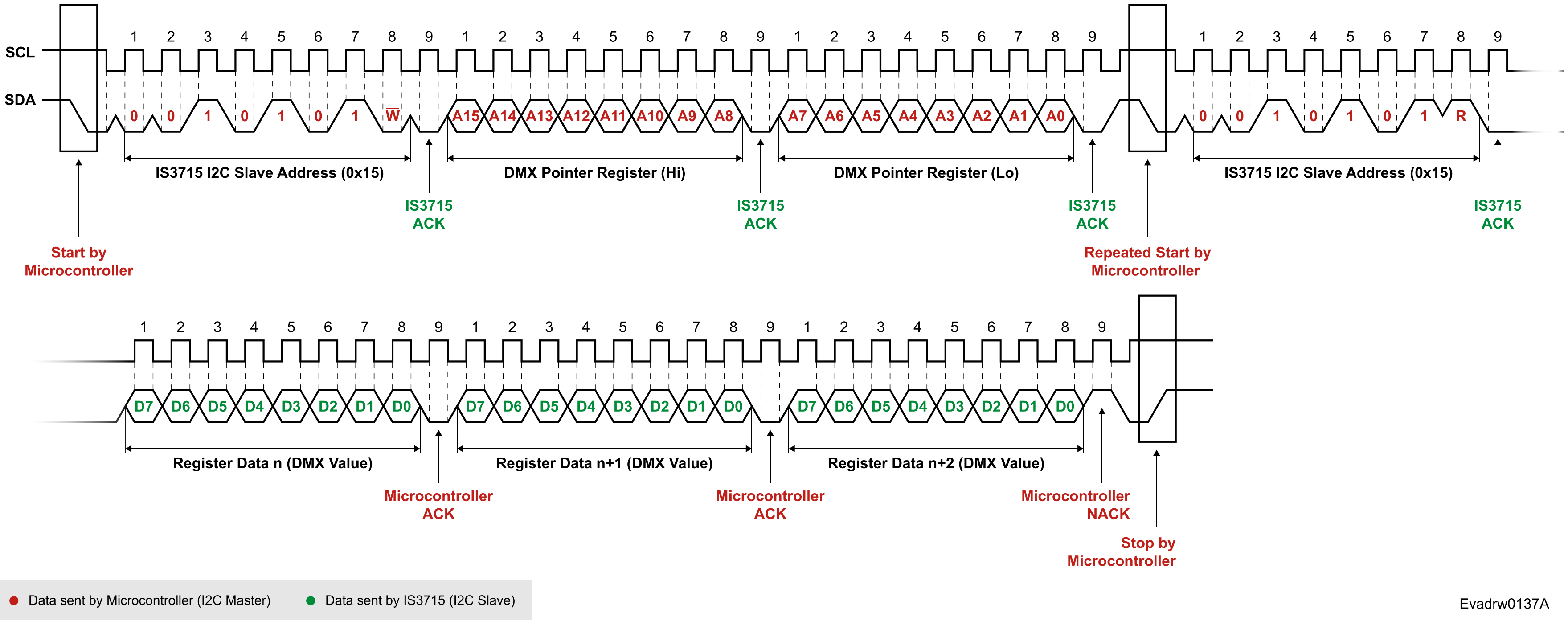
With each byte read, the DMX Pointer Register increments by one.

Invalid Memory Addressing

The valid memory range of the IS3715 goes from addresses `0` to `514`.

If the Read Operation is performed with a Pointer Register within the valid memory range (`0` to `514`), but the data retrieval extends beyond register `514`, a rollover to position `0` will occur. For example, the value of register `516` will correspond to the content of register `1` (`DMX_CH1`).

If a Read Operation is performed with a Pointer Register value higher than `514`, the read result will be `0xFF`.

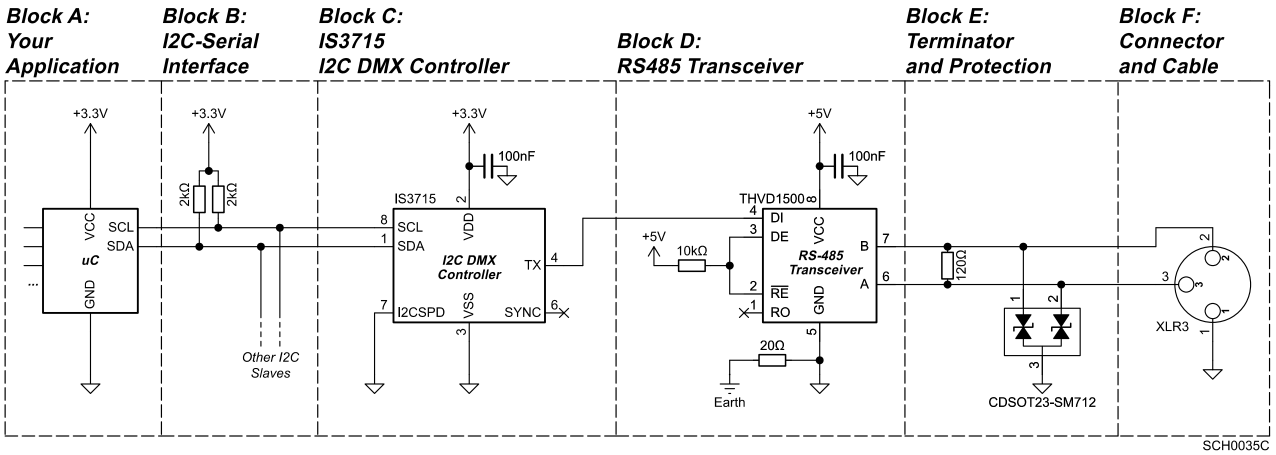

SCL
SDA
1 2 3 4 5 6 7 8 9
0 0 1 0 1 0 1 W̄
A15 A14 A13 A12 A11 A10 A9 A8
A7 A6 A5 A4 A3 A2 A1 A0
0 0 1 0 1 0 1 R
IS3715 I2C Slave Address (0x15)
DMX Pointer Register (Hi)
DMX Pointer Register (Lo)
IS3715 I2C Slave Address (0x15)
IS3715 ACK
IS3715 ACK
IS3715 ACK
IS3715 ACK
Start by Microcontroller
Repeated Start by Microcontroller
D7 D6 D5 D4 D3 D2 D1 D0
Register Data n (DMX Value)
Register Data n+1 (DMX Value)
Register Data n+2 (DMX Value)
Microcontroller ACK
Microcontroller ACK
Microcontroller NACK
Stop by Microcontroller
● Data sent by Microcontroller (I2C Master)
● Data sent by IS3715 (I2C Slave)
Evadrw0137A

6 Hardware Examples

The following chapter represents an application design example for explanation proposals and is not part of the product standard. The customer must design his own solution, choose its most appropriate components and validate the final product according to the legislation and the DMX512 specifications.

6.1 DMX Controller

This example shows the design of a standard non-isolated DMX Controller.



Block A: Your Application

This is usually the main part of your product, where you read sensors, potentiometers, or acquire data that needs to be converted to DMX.

Typically, a microcontroller interfaces with the IS3715, but a microprocessor or a single-board computer, such as a Raspberry Pi, can also be used as long as they are equipped with an I2C-Serial Interface.

Block B: I2C-Serial Interface

For proper operation of the I2C-Serial Interface, pull-up resistors to 3.3 V or 5 V are necessary. Typical resistor values are 4.7 kΩ for Standard Mode (100 kHz) and 2 kΩ for both Fast Mode (400 kHz) and Fast Mode Plus (1 MHz).

Block C: IS3715

The IS3715 is very simple to integrate into your design.

A decoupling capacitor should be placed on the power pins (VDD and VSS). It is recommended to use a 100 nF ceramic capacitor.

The I2CSPD pin defines the I2C speed. Connect this pin to GND for a speed of 100 kHz. For 400 kHz, it should be pulled to 1.65 V, which is half of 3.3 V. This can be achieved with a simple resistor voltage divider using 3.3 V and GND. For 1 MHz, the pin must be connected to 3.3 V. This pin is not 5 V tolerant.

Block D: Transceiver

DMX operates over the RS485 electrical standard. Therefore, an RS485 transceiver or driver is required to convert the TTL-compatible voltage levels to differential RS485 signals before sending them to the DMX bus.

Since a DMX controller never receives data, the DE and RE pins of the transceiver can be tied to VCC to keep it always in transmit mode.

Even if the transceiver is only used as a driver and therefore the receiver part will never be used, it is sometimes preferred because it can be cheaper than a receiver-only device. Compare distributor prices to validate your preferred option.

Either 3.3 V or 5 V transceivers and drivers can be used, but 5 V ones are preferred, as they offer better noise immunity on the DMX bus.

Isolation

Isolation is a complex topic, especially in long cable runs, where ground potentials can differ significantly, or when all the equipment is not powered from the same source, e.g., mixing mains line and a gasoline generator.

This example is a Ground Referenced Transmitter (non-isolated), which is the recommended by the DMX512-A standard.

For a non-isolated transmitter, the standard does not require product labeling. For an isolated transmitter, the standard requires it to be labeled as either “ISO” or “ISOLATED”.

Block E: Terminator and Protection

Terminator

Reflections on a transmission line occur whenever there’s an impedance mismatch that a traveling wave encounters as it moves along the line. To reduce reflections at the ends of a DMX cable, a line termination should be placed near each end of the bus. Terminating both ends is crucial because signals travel in both directions, but no more than two terminators should be used on the same bus. The line terminator connects across the balanced lines (cable A and B) and is a 120 Ω resistor rated at 1 W.

On the controller, which normally only has a DMX Out connector, the terminator resistor is usually integrated on the PCB. On the receivers, it is not integrated, as they have a DMX Out connector to daisy-chain another receiver. Therefore, on the receivers, you terminate the line by placing a special terminator connector, which contains nothing but a 120 Ω resistor.

Protection

The protection stage is influenced by several factors, including the intrinsic robustness and protection features of the transceiver or driver chip, the product's budget, and its required reliability, among other considerations. Refer to your transceiver's documentation to determine the appropriate protection requirements.

In the schematic, a bidirectional 400-W transient suppressor diodes (CDSOT23-SM712) are used to protect against surge transients.

Block F: Connector and Cable

Connector

The official DMX connector is the XLR-5. Exceptions include RJ45, miniature connectors, and screw terminal connectors. However, despite its popularity and widespread use, XLR-3 is not part of the DMX standard and should not be used. But what happens in the real world and in this example?

Generally, XLR-3 connectors are cheaper than XLR-5 connectors. Therefore, XLR-5 connectors are typically found in professional equipment, while XLR-3 connectors are more common in cost-sensitive devices.

In this example, an XLR-3 connector has been used due to its widespread popularity and clarity of implementation, but we strongly encourage product designers to follow the standard and use XLR-5 connectors.

Using an XLR-3 connector has the drawback of making your product compatible with standard microphone cables, which are specifically designed for low-frequency analog audio—not digital signals. As a result, microphone cables are not suitable for DMX, as they degrade the DMX signal, reducing the maximum cable length and increasing the chances of flickering.

DMX controllers usually only feature a DMX Out connector (female), while DMX receivers have both DMX In (male) and DMX Out (female) connectors for daisy-chaining—that is, two DMX connectors.

- DMX Out: Female connector
- DMX In: Male connector

In both XLR-3 and XLR-5, and in both male and female connectors, the pinout is as follows:

- Pin 1: Singal-Common, this connects to the cable screen.
- Pin 2: Data – (Also called ‘B’)
- Pin 3: Data + (Also called ‘A’)

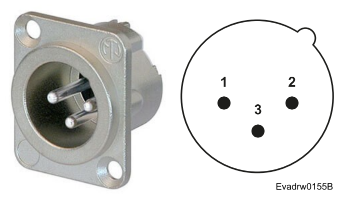
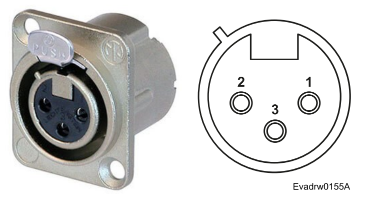
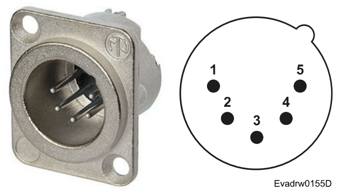
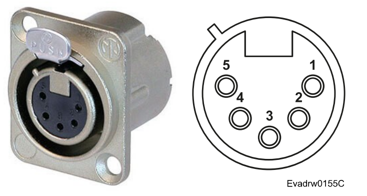
Cable

The DMX cable screen must be connected to the pin 1 of the XLR-3 or XLR-5 connector and not to its shell. Do not connect the cable screen to the connector shell.

Use only twister pair cable to carry the DMX signal.

Do not use microphone cable, as it has been designed to carry low-frequency signals, and it will degrade the DMX signal, increasing the chances of spurious flickers on the LED fixtures.

6.2 DMX Pinout

	DMX In (Male)	DMX Out (Female)	Pinout (Male and Female)
XLR-3			• Pin 1: Signal Common • Pin 2: Data – (B) • Pin 3: Data + (A)
XLR-5			

7 Firmware Examples

7.1 Arduino Code Example

Example Reference Code
ISXMPL3715ex2

Objective of this example
Demonstrate how to write a program for the IS3715 I2C DMX Controller using Arduino.

Required Material

- Kappa3715Ard Board (INACKS evaluation board featuring the IS3715)
- Arduino UNO Board
- DMX-compatible light fixture

Setup
Flash the Arduino UNO with the provided code, connect the Kappa3715 to the Arduino UNO, and attach a DMX light to validate its operation.

How it works
The Arduino code reads the analog value from the potentiometer on the Kappa3715Ard board. This reading is scaled to a value from 0 to 255, which corresponds to the valid range for a DMX channel. The scaled value is then written to IS3715 register 1, which controls DMX Channel 1.

Other
Download the example code at: https://inacks.com/is3715

Find Kappa3715Ard product information at: https://inacks.com/kappa3715ard

```
/*
 * This example works with the Evaluation Board Kappa3715,
 * which features an IS3715-I2C DMX Controller chip.
 * The Arduino reads a potentiometer connected to 3.3 V on pin A0,
 * scales the value to a DMX range (0-255), and writes it to the IS3715,
 * which continuously transmits DMX data.
 * For more information, visit www.inacks.com
 */
#include <Wire.h>

// IS3715 Memory Map Registers:
#define DMX_CH1    1
#define DMX_CH2    2
#define DMX_CH3    3
// ...
#define DMX_CH512 512
#define CHIP_ID   513
#define CHIP_REV  514

// Constants:
#define I2C_DEVICE_ADDRESS  21
#define CHIP_ID_VALUE       21

// This routine writes a dmxValue to the specified memory address of
// the IS3715 memory map.
void writeIS3715Register(uint16_t address, uint8_t dmxValue) {
  Wire.beginTransmission(I2C_DEVICE_ADDRESS);

  // Send the 16-bit memory address (high byte first, then low byte).
  Wire.write((address >> 8) & 0xFF);
  Wire.write(address & 0xFF);

  // Send the 8-bit data value
  Wire.write(dmxValue);

  Wire.endTransmission();
}

// Read one byte from the specified register address in the IS3715 memory map.
uint8_t readIS3715Register(uint16_t holdingRegisterAddress) {
  uint8_t result; // Variable to store the read data.
  Wire.beginTransmission(I2C_DEVICE_ADDRESS);

  // Send the 16-bit memory address (high byte first, then low byte).
  Wire.write((holdingRegisterAddress >> 8) & 0xFF);
  Wire.write(holdingRegisterAddress & 0xFF);
  // Send a repeated START condition (no STOP)
  Wire.endTransmission(false);
  // Request 1 byte from the IS3715.
  Wire.requestFrom(I2C_DEVICE_ADDRESS, 1);
  result = Wire.read();
  return result;
}

void setup() {
  uint16_t chipID, chipRev;

  // Initialize the I2C interface
  Wire.begin();
  // Initialize the serial port for debug output
  Serial.begin(9600);

  // Detect the chip:
  chipID = readIS3715Register(CHIP_ID);
  chipRev = readIS3715Register(CHIP_REV);
  if (chipID == CHIP_ID_VALUE) {
    Serial.println("IS3715 Chip detected on I2C!");
    Serial.print("Chip ID: "); Serial.println(chipID);
    Serial.print("Chip Rev: "); Serial.println(chipRev);
  }
  else {
    Serial.print("ERROR: IS3715 Chip NOT detected on I2C!");
  }
}

void loop() {
  // Read the potentiometer (0-3.3V) on analog pin A0
  uint16_t potValue = analogRead(A0);

  // Scale the analog reading to 0-255 (DMX value range)
  uint8_t dmxValue = map(potValue, 0, 678, 0, 255);
  Serial.print("Potentiometer value: ");
  Serial.println(dmxValue);

  // Write the scaled DMX value into DMX channel 1 register
  writeIS3715Register(DMX_CH1, dmxValue);
}
```

7.2 STM32 Code Example

Example Reference Code
ISXMPL3715ex3

Objective of this example
Demonstrate how to write a program for the IS3715 I2C DMX Controller using STM32 Cube IDE.

Required Material
- Kappa3715Ard Board (INACKS evaluation board featuring the IS3715)
- STM32 Nucleo-C071RB
- DMX-compatible light fixture

Setup
Download the ISXMPL3715ex3 from https://inacks.com/is3715 and flash it to the Nucleo-C071RB Board, connect the Kappa3715 to the Nucleo-C071RB, and attach a DMX light to validate its operation.

How it works
The CubeIDE code continuously increments the variable `dmxChannel1Value` by 1. This value is written to register address 1 of the IS3715, which causes the DMX channel value to increase continuously.

Other
Download the example code at: https://inacks.com/is3715

Find Kappa3715Ard product information at: https://inacks.com/kappa3715ard

Attention
The following code is an extraction of the relevant parts of the CubeIDE project.

Pasting this code directly into CubeIDE will not work, as all the CubeIDE libraries and extra functions from its HAL libraries are omitted for pedagogical purposes.

Download the full CubeIDE project or paste this code into the appropriate sections of a CubeIDE project.

```
#include <stdio.h>

// IS3715 Memory Map Registers:
#define CHIP_ID       513
#define CHIP_REV      514

// This variable stores the data of the DMX Channel 1:
uint8_t dmxChannel1Value;

/**
 * @brief     This routine reads a single register from the IS3715 memory map.
 * @param     registerAdressToRead: Address in the IS3715 memory map to be read.
 * @retval    Value stored at the registerAdressToRead register address.
 */
uint8_t readIS3715Register(uint16_t registerAdressToRead) {
       uint8_t readResult = 0;
       uint8_t IS3715_I2C_Chip_Address;
    IS3715_I2C_Chip_Address = 0x15; // IS3715 I2C address is 0x15 (7-bit).
    // The STM32 HAL I2C library requires the I2C address to be shifted left by one bit.
    // Let's shift the IS3715 I2C address accordingly:
    IS3715_I2C_Chip_Address = IS3715_I2C_Chip_Address << 1;

    HAL_I2C_Mem_Read(&hi2c1, IS3715_I2C_Chip_Address, registerAdressToRead, I2C_MEMADD_SIZE_16BIT,
&readResult, 1, 500);

    return readResult;
}

/**
 * @brief     Reads a single register from the IS3715 memory map.
 * @param     registerAdressToRead: Address in the IS3715 memory map to be read.
 * @retval    Value stored at the registerAdressToRead register address.
 */
void writeIS3715Register(uint16_t registerAdressToWrite, uint8_t value) {
    uint8_t IS3715_I2C_Chip_Address;  // I2C address of IS3715 chip (7-bit).
    IS3715_I2C_Chip_Address = 0x15; // IS3715 I2C address is 0x15 (7-bit).
    // STM32 HAL expects 8-bit address, so shift left by 1:
    IS3715_I2C_Chip_Address = IS3715_I2C_Chip_Address << 1;

    HAL_I2C_Mem_Write(&hi2c1, IS3715_I2C_Chip_Address, registerAdressToWrite,
I2C_MEMADD_SIZE_16BIT, &value, 1, 500);
}

// This function is required to make printf work over the Serial port.
// It is never called directly in the code — printf internally uses it
int _write(int file, char *ptr, int len) {
       HAL_UART_Transmit(&huart2, (uint8_t*)ptr, len, HAL_MAX_DELAY);
    return len;
}

int main(void) {
       HAL_StatusTypeDef chipDetected = HAL_I2C_IsDeviceReady(&hi2c1, (0x15 << 1), 3, 200);

       if (chipDetected == HAL_OK) {
         printf("IS3715 Detected!\n");
       }
       else {
         printf("ERROR: IS3715 not detected. Program stops now.\n");
         while(1);
       }

       while (1) {
         writeIS3715Register(1, dmxChannel1Value);
         dmxChannel1Value++;
         HAL_Delay(10);
       }
}
```

7.3 Raspberry Pi Python Example

Example Reference Code
ISXMPL3715ex4

Objective of this example
Demonstrate how to write a program in Python for the IS3715 I2C DMX Controller in Raspberry Pi.

Required Material
- Kappa3715Rasp Board (INACKS evaluation board featuring the IS3715)
- Raspberry Pi B Board
- DMX-compatible light fixture

Setup
Connect the Kappa371Rasp to the Raspberry Pi, and attach a DMX light to the Kappa371Rasp. Run the command `sudo i2cdetect -y 1` to verify that the IS3715 is detected. Finally, execute example with `sudo`.

How it works
The Python code continuously increments the `DMX_Values[1]` by `1`. This value is written to register address 1 of the IS3715, which causes the DMX channel value to increase continuously.

Other
Download the example code at: https://inacks.com/is3715

Find Kappa3715Ard product information at: https://inacks.com/kappa3715rasp

Example of chip detection on the I2C-Serial interface
To verify that the Raspberry Pi is properly connected to the IS3715, scan the I2C-Serial interface for connected devices.

The IS3715 should appear at address 21 (0x15)

Run:

```
sudo i2cdetect -y 1
```

The command output should show a detected device at 21 (0x15)

```
pi@raspberrypi:~ $ sudo i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- 15 -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

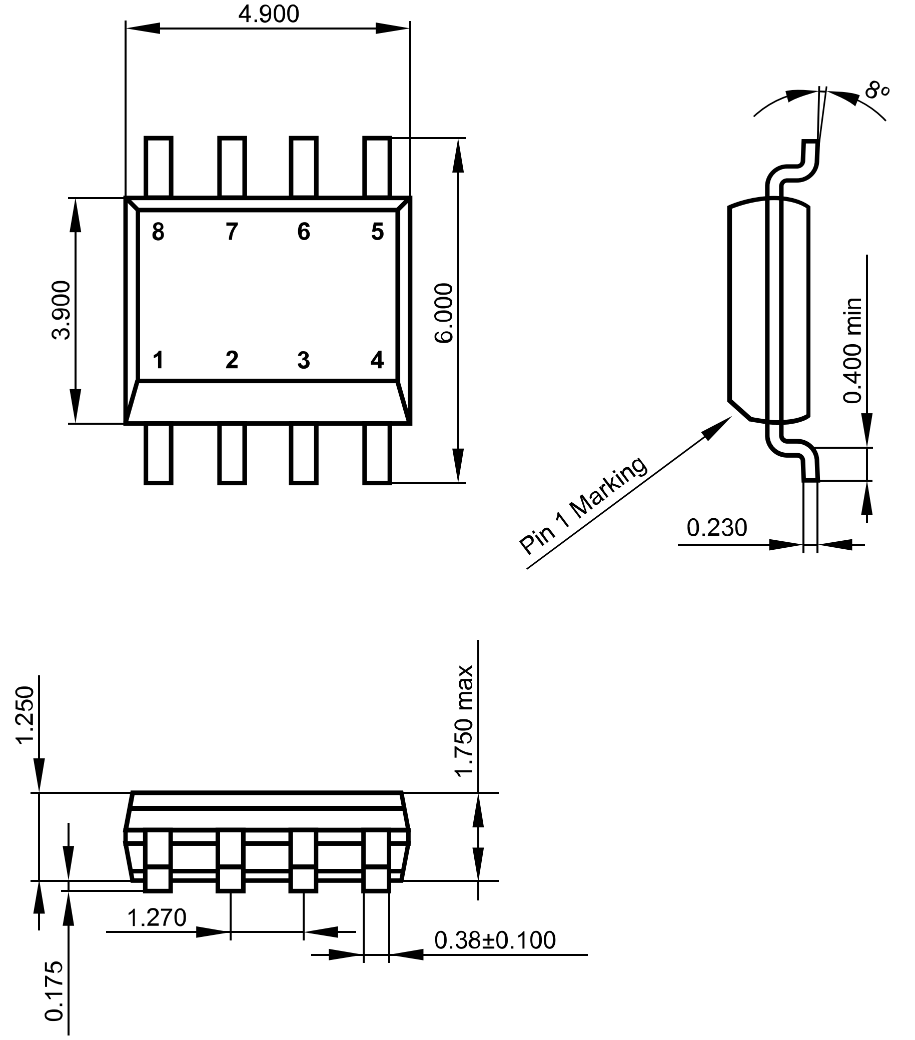
Python Code

```python
"""
This example continuously increments the value of DMX Channel 1.
When the value reaches 255, it rolls over to 0.

This example uses the Kappa3715 Evaluation Board featuring the IS3715.

Verify that your Raspberry Pi can see the IS3715 on the I2C Serial Interface:
$ sudo i2cdetect -y 1
The device should appear at address 0x15.

Run this script with sudo:
$ sudo python ISXMPL3715ex4-IS3715_RaspberryPi_Example.py

For more information, visit www.inacks.com
"""

from smbus2 import SMBus, i2c_msg
import time

I2C_BUS = 1  # Use 1 for most Raspberry Pi models
I2C_DEVICE_ADDRESS = 21  # I2C device address of the IS3715
```

```
# IS3715 Memory Map Addresses:
CHIP_ID = 513
CHIP_REV = 514

def write_register(start_register, data_bytes):
    """
    Write a block of data starting at a 16-bit register address.

    :param start_register: The 16-bit register address to start writing to.
    :param data_bytes: A list of bytes to write.
    """
    high_addr = (start_register >> 8) & 0xFF # High byte of 16-bit address
    low_addr = start_register & 0xFF # Low byte of 16-bit address
    with SMBus(I2C_BUS) as bus:
        # Send 16-bit address followed by data bytes
        msg = i2c_msg.write(I2C_DEVICE_ADDRESS, [high_addr, low_addr] + data_bytes)
        bus.i2c_rdwr(msg)

def read_registers(start_register, length):
    """
    Read a block of data starting at a 16-bit register address.

    :param start_register: The 16-bit register address to start reading from.
    :param length: Number of bytes to read.
    :return: A list of bytes read from the device.
    """
    high_addr = (start_register >> 8) & 0xFF # High byte of 16-bit address
    low_addr = start_register & 0xFF # Low byte of 16-bit address
    with SMBus(I2C_BUS) as bus:
         # First send the 16-bit register address (write without stop / repeated start)
        write_msg = i2c_msg.write(I2C_DEVICE_ADDRESS, [high_addr, low_addr])
        # Then read the requested number of bytes
        read_msg = i2c_msg.read(I2C_DEVICE_ADDRESS, length)
        bus.i2c_rdwr(write_msg, read_msg)
        return list(read_msg)


# Detect the IS3715 on the I2C bus
print("Reading CHIP_ID register...")
chip_id_value = read_registers(CHIP_ID, 1) # Read the chip ID
chip_rev_value = read_registers(CHIP_REV, 1) # Read the chip revision
print "CHIP_ID: ", chip_id_value
print "CHIP_REV: ", chip_rev_value

# Verify if the chip ID matches expected value (21)
if chip_id_value[0] == 21:
    print("IS3715 Detected!")
else:
    print("ERROR: IS3715 not detected!")

# Initialize the DMX array
DMX_Values = [0] * 513 # 0 to 512
DMX_Channel = 1 # We will increment channel 1

while True:
    # Print the current DMX value being written to channel 1
    print "Writing on DMX Channel", DMX_Channel, "value", DMX_Values[1]

    # Write the entire DMX memory map starting from start code (address 0)
    write_register(0, DMX_Values)

    # Increment the DMX channel value, rollover after 255
    DMX_Values[DMX_Channel] = DMX_Values[DMX_Channel] + 1
    if DMX_Values[DMX_Channel] > 255:
        DMX_Values[DMX_Channel] = 0
```

8 Mechanical

SO8N Package

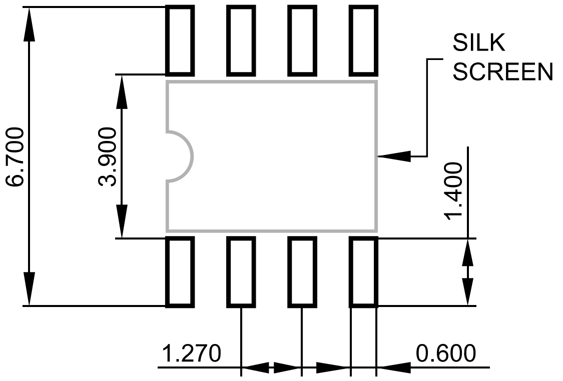


Units: millimeters

Notes:
This drawing is for general information only.
Drawing not to scale.

Evadrw0033A

SO8N Recommended Footprint



Units: millimeters

Notes:
This drawing is for general information only.
Drawing not to scale.

Evadrw0025r2

Content

Appendix

Revision History

Document Revision

Date	Revision Code	Description
September 2025	ISDOC143**A**	- Initial Release

Chip Revision

Chip Revision can be found in the `CHIP_REV` register of the memory map.

Date	Revision Code	Description
September 2025	**0**	- Initial Release

Documentation Feedback

Feedback and error reporting on this document are very much appreciated. Please indicate the code or title of the document.

feedback@inacks.com

Sales Contact

For special order requirements, large volume orders, or scheduled orders, please contact our sales department at:

sales@inacks.com

Customization

INACKS can develop new products or customize existing ones to meet specific client needs. Please contact our engineering department at:

engineering@inacks.com

Trademarks

This company and its products are developed independently and are not affiliated with, endorsed by, or associated with any official protocol or standardization entity. All trademarks, names, and references to specific protocols remain the property of their respective owners.

Disclaimer

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, INACKS does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. INACKS takes no responsibility for the content in this document if provided by an information source outside of INACKS.

In no event shall INACKS be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, INACKS's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of INACKS.

Right to make changes — INACKS reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — INACKS products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an INACKS product can reasonably be expected to result in personal injury, death or severe property or environmental damage. INACKS and its suppliers accept no liability for inclusion and/or use of INACKS products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Quick reference data — The Quick reference data is an extract of the product data given in the Limiting values and Characteristics sections of this document, and as such is not complete, exhaustive or legally binding.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. INACKS makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using INACKS products, and INACKS accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the INACKS product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

INACKS does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using INACKS products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). INACKS does not accept any liability in this respect.

Limiting values — Stress above one or more limiting values (as defined in the Absolute Maximum Ratings System of IEC 60134) will cause permanent damage to the device. Limiting values are stress ratings only and (proper) operation of the device at these or any other conditions above those given in the Recommended operating conditions section (if present) or the Characteristics sections of this document is not warranted. Constant or repeated exposure to limiting values will permanently and irreversibly affect the quality and reliability of the device.

Terms and conditions of commercial sale — INACKS products are sold subject to the general terms and conditions of commercial sale, as published at http://www.inacks.com/comercialsaleterms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. INACKS hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of INACKS products by customer.

No offer to sell or license — Nothing in this document may be interpreted or construed as an offer to sell products that is open for acceptance or the grant, conveyance or implication of any license under any copyrights, patents or other industrial or intellectual property rights.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Non-automotive qualified products — This INACKS product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. INACKS accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

Protocol Guidance Disclaimer: The information provided herein regarding the protocol is intended for guidance purposes only. While INACKS strive to provide accurate and up-to-date information, this content should not be considered a substitute for official protocol documentation. It is the responsibility of the client to consult and adhere to the official protocol documentation when designing or implementing systems based on this protocol.

INACKS make no representations or warranties, either expressed or implied, as to the accuracy, completeness, or reliability of the information contained in this document. INACKS shall not be held liable for any errors, omissions, or inaccuracies in the information or for any user's reliance on the information.

The client is solely responsible for verifying the suitability and compliance of the provided information with the official protocol standards and for ensuring that their implementation or usage of the protocol meets all required specifications and regulations. Any reliance on the information provided is strictly at the user's own risk.

Certification and Compliance Disclaimer: Please be advised that the product described herein has not been certified by any competent authority or organization responsible for protocol standards. INACKS do not guarantee that the chip meets any specific protocol compliance or certification standards.

It is the responsibility of the client to ensure that the final product incorporating this product is tested and certified according to the relevant protocol standards before use or commercialization. The certification process may result in the product passing or failing to meet these standards, and the outcome of such certification tests is beyond our control.

INACKS disclaim any liability for non-compliance with protocol standards and certification failures. The client acknowledges and agrees that they bear sole responsibility for any legal, compliance, or technical issues that arise due to the use of this product in their products, including but not limited to the acquisition of necessary protocol certifications.