IS3720: I2C DMX+RDM Receiver IC

512 Channels DMX Receiver + RDM Responder In a Single Chip

Applications

- Professional Light Fixtures
- Architectural & Building Lighting
- Stage & Entertainment Lighting
- Facade Lighting
- In-Ground Lighting
- Ornamental Fountain Lighing
- Difficult-Access Lighting
- Digital Art Installations
- Animatronics
- OEM / Device Manufacturers

Main Advantages

- Reduces engineering time and costs.
- Reduces product time-to-market.
- Makes the DMX and RDM 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, UART, etc.).
- Compact, easy-to-solder SO8N package.

DMX, RDM and I2C Characteristics

- Receives from 1 up to all 512 DMX Channels.
- Implemented RDM PIDs:
  - Required PIDs: All (ANSI E1.20-2010)
  - Other PIDs:
    - DEVICE_MODEL_DESCRIPTION
    - MANUFACTURER_LABEL
    - SOFTWARE_VERSION_LABEL
- RDM parameters are stored in non-volatile memory.
- Easy UID programming for mass device production.
- I2C speeds: 100 kHz, 400 kHz, and 1 MHz.

General Description

The IS3720 is a chip that combines DMX Receiver + RDM Responder protocols in a single device. It continuously buffers all 512 DMX channels in its memory map, which is accessible by a microcontroller via I2C, and enables bidirectional communications with RDM.

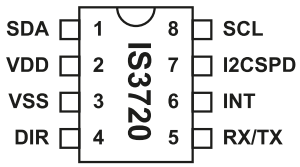
Having RDM provides a multitude of benefits, such as remotely setting its DMX address and automatic patching, making it ideal for large installations and difficult-to-access fixtures, such as architectural or underwater lighting.

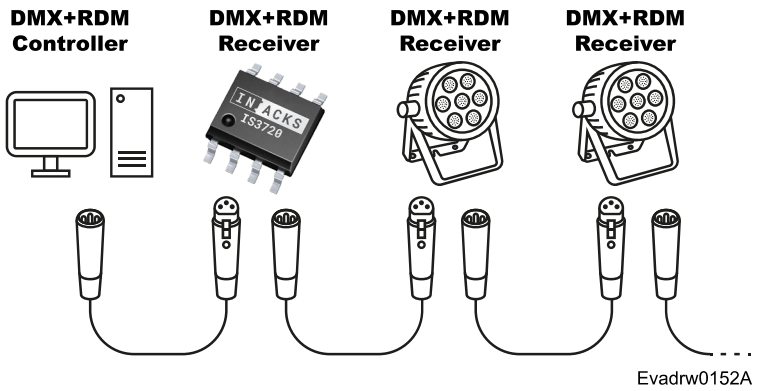
The IS3720 RDM parameters can be fully customized for your brand. Therefore, when a RDM controller reads your fixture, it will display your company and fixture model names.

The IS3720 frees your microcontroller from the DMX and RDM protocol timing constraints, memory requirements, and ISR load associated with reading and generating DMX and RDM data, reducing the need for timers, flash, RAM, and dedicated pins. All these characteristics help reduce the engineering costs and time required to successfully launch your product to the market.

The device operates at 3.3 V, with 5 V-tolerant I2C and UART, enabling compatibility with 3.3 V or 5 V microcontrollers and RS485 transceivers.

Easy-to-solder SO8N (4.9×6 mm) package, supporting an industrial temperature range (-40 to 85 °C). Special on-demand packages and temperature ranges are also available: UFQFPN 3×3 mm, WLCSP 1.7×1.42 mm; extended (-40 to 105 °C) and high (-40 to 125 °C) temperature ranges.





1 Specifications

Absolute Maximum Ratings

Parameter				Min	Max	Unit
Input Voltage	VDD pin			-0.3	4	V
	SCL, SDA, RX/TX, DIR pins			-0.3	6	
	I2CSPD pin			-0.3	4	
Current Sourced/Sunk by any I/O or Control Pin				-	±20	mA
Current Sourced/Sunk by sum of all I/O or Control Pin				-	±80	
Temperature	Ambient Temperature (T_A) / [Junction Temperature (T_J)]	IS3720-S8-I	Industrial	-40	85 [105]	ºC
		IS3720-S8-E	Extended	-40	105 [125]	
		IS3720-S8-H	High	-40	125 [130]	
	Storage Temperature			-65	150	
Internal Non-Volatile Memory	Endurance[1]	T_J = -40 to 130 ºC		10	-	kcycles
	Data retention[1]	1 kcycle at T_A = 85 ºC		30	-	Years

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, RX/TX, DIR 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, RX/TX, DIR pins	$I_{I/O-SS}$	-	±6	±15	mA

Electrical Characteristics

Parameter		Symbol	Min	Typ	Max	Unit
Current Consumption (T_A = 85ºC)		I_{OP}	-	3.50	4.10	mA
Input Voltage	Logical High-Level	V_{IH}	0.7xV_{DD}	-	-	V
	Logical Low-Level	V_{IL}	-	-	0.3xV_{DD}	

Electrical Specifications Revision B

1 Evaluated by characterization. Not tested in production.

RDM and DMX Timing

Parameter		Symbol	Min	Nom	Max	Unit
Break	RDM Transmitting	T_{BREAK}	-	243	-	µs
	DMX Receiving		-	108	-	
Mark After Break	RDM Transmitting	T_{MAB}	-	22	-	
	RDM Receiving		-	12	-	
Data Bit	RDM Transmitting	T_{BIT}	3.98	-	4.02	
Slot	RDM Transmitting	T_{SLOT}	-	44.06	-	
Controller: RDM Request → Responder: RDM Response		T_{RR}	-	240	-	
Controller: RDM Discovery → Responder: RDM Response		T_{DR}	-	210	-	

RDM Supported Parameters ID (PIDs)

RDM Parameter IDs (PIDs)	Code	Supported
Category – Network Management		
DISC_UNIQUE_BRANCH	0x0001	✓
DISC_MUTE	0x0002	✓
DISC_UN_MUTE	0x0003	✓
Category – RDM Information		
SUPPORTED_PARAMETERS	0x0050	✓
PARAMETER_DESCRIPTION	0x0051	✓
Category – Product Information		
DEVICE_INFO	0x0060	✓
DEVICE_MODEL_DESCRIPTION	0x0080	✓
MANUFACTURER_LABEL	0x0081	✓
SOFTWARE_VERSION_LABEL	0x00C0	✓
Category – DMX512 Setup		
DMX_START_ADDRESS	0x00F0	✓
Category – Control		
IDENTIFY_DEVICE	0x1000	✓

2 Description

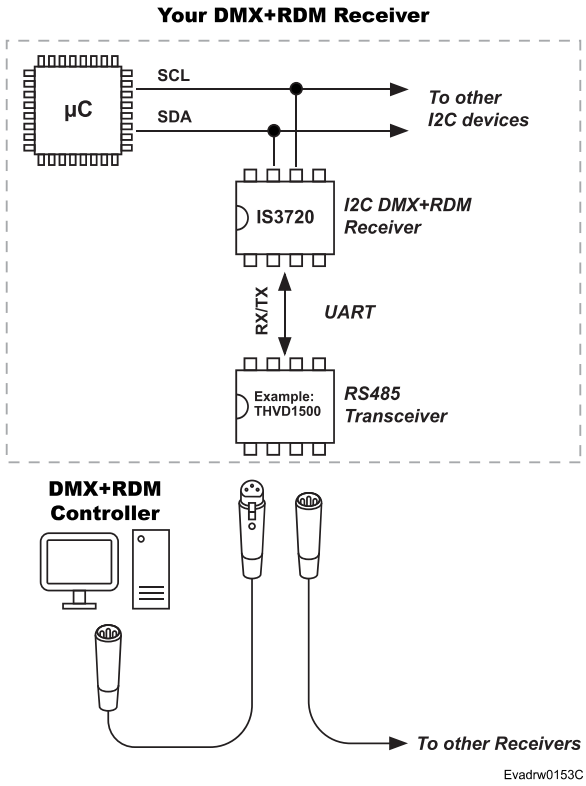
2.1 IS3720 Description

What is it and what it does
The IS3720 is a 3.3 V operating chip that integrates both DMX Receiver + RDM Responder protocols in a single IC. It buffers all 512 DMX channels in its internal memory map, which is accessible by your microcontroller via I2C.

The IS3720 also enables bidirectional communications through RDM, allowing remote configuration and management of your product. This eliminates the need for manual setup, such as physically adjusting the DMX start address.

Utility
RDM is especially helpful for light fixtures in large installations, such as building façade lighting, where an operator would otherwise spend significant time manually setting DIP switches on thousands of fixtures distributed across multiple floors and rooms.

It is also invaluable in installations where fixtures are embedded in the pavement or other sealed environments, like swimming pools or ornamental fountain where opening them would require breaking the ground or compromising waterproofing.



Benefits
The IS3720 eliminates the need for engineers to learn and implement the low-level details of the DMX and RDM protocols. This significantly reduces development and testing effort, shortens time-to-market, and enables faster product release.

How it works
The IS3720 features two communication interfaces:

- An I2C-serial interface (SCL and SDA pins), operating as slave device (address 22), used to communicate with your microcontroller.
- A UART interface (RX/TX and DIR pins), used for DMX+RDM communications.

The UART pins operate at TTL voltage levels, while the DMX network uses RS485 voltage levels. Therefore, an external RS485 transceiver, such as the THVD1500, is required to convert between TTL and RS485 levels. The UART pins are 5 V-tolerant. Additional design details can be found in the Hardware Examples section.

The IS3720 has an internal memory map consisting of 632 registers, each 8 bits wide, with addresses ranging from `0` to `631`. Therefore, accessing any register requires a 2-byte addressing.

Registers `1` to `512` correspond directly to DMX channel values. For example, reading register address `1` gets the value of DMX Channel 1, reading register address `512` reads the value of DMX Channel 512, covering the full range of channels.

RDM registers are mapped after the DMX registers. Some are informational, such as the MANUFACTURER_LBL register block, where you write via I2C your company name and it will show up on the RDM controller. Others are operational, such as the DMX_START_ADDRESS register block, which indicates the assigned DMX start address to the microcontroller.

2.2 Provisioning

Once the product has been finalized and validated, it enters volume manufacturing. During this phase, RDM parameters—such as the manufacturer name, product name and number, and software name and version—can be easily configured by the firmware via I2C at startup, without requiring any special operations, as these values remain constant across the entire product line. The non-volatile memory is only updated if the written values differ from the previous ones.

The UID, however, must be unique for each manufactured unit, and configuring it via I2C can be more complex.

To simplify UID configuration on the production line, a special PID (0x8000), called “Update UID”, becomes available when writing the key value `160` to the RDM_ONLINE register. Writing a new UID to this PID via RDM enables efficient volume manufacturing while ensuring a unique UID is assigned to each device.

3 Usage

The IS3720 is very simple to use. It has been designed to minimize the time required to implement the DMX+RDM protocol.

Your microcontroller only needs to configure the RDM registers at startup and, once done, retrieve the desired DMX Channels.

3.1 Example: Designing an underwater RGB light with RDM

During startup, perform the following steps:

1. **Write the Unique Identification Number** (6 bytes) of the device to the memory map register block `UID`.

   This number uniquely identifies your RDM product from all other RDM products on the market.

   For prototyping, leave the default `UID` and skip this step.

   Refer to the `UID` section for more information on how to acquire a UID.

   When the UID data changes, it is automatically stored in the IS3720's internal non-volatile memory and reloaded each time the chip powers up.

2. **Write the category code** (2 byes) of the device to the `CATEGORY` register block in the memory map.
   In this example, a fixed RGB light corresponds to `0x01`, `0x01`: PRODUCT_CATEGORY_FIXTURE_FIXED.

   This information defines the type of product your device is, such as a light, fog machine, projector, dimmer, etc.

   It does not affect the operation of the device but helps the lighting operator perform their tasks.

   The values of this register are standardized and recognized by all RDM controllers, according to Table 2: RDM Standardized Product Categories.

3. **Write the DMX footprint** (2 bytes) of the device to the `DMX_FOOTPRINT` register block in the memory map.
   In this example, a fixed RGB light corresponds to `0x00`, `0x03`, as it uses three DMX channels.

   This information tells the lighting operator how many channels the device needs.

   This register is especially useful because it enables the autopatch feature in RDM controllers, making setup much faster, particularly in large DMX installations.

4. Optionally, you can **write your company name**, product model number and name, software version and name by writing to the `MANUFACTURER_LBL`, `MODEL_NUM`, `MODEL_LBL`, `SOFT_NUM`, `SOFT_LBL` register blocks in the memory map.

   This information doesn't affect how your product works but helps the lighting operator identify your device.

5. You're ready! **Bring the RDM device online** by writing a 1 to the `RDM_ONLINE` register.

In the main loop of your code, perform the following steps:

6. **Keep monitoring `DMX_START_ADDRESS` and `IDENTIFY_DEVICE`** register by reading its values or monitoring changes on the IS3720's INT pin.

   Occasionally, the lighting operator may change the DMX_START_ADDRESS of your device, usually when an underwater light is first installed, or during a stage setup for a show.

   During these initial tasks, the operator may set the IDENTIFY_DEVICE register to `1`, which tells your device to "show itself" overriding the current DMX data it is receiving. This is very useful, for example, in our underwater scenario, it helps to identify a specific light among the rest of the lights.

7. **Keep reading the DMX data** assigned to the light and dim the RGB LEDs accordingly.

   The IS3720 stores all DMX data in its DMX_CHx registers, but you should read only the channels assigned to your device. Reading all the DMX_CHx registers takes more time and can reduce the light's refresh rate.

   The DMX_START_ADDRESS register tells you in which DMX_CHx start reading from, and the DMX_FOOTPRINT register tells you how many channels to read.

Continuously perform steps 6 and 7 in your main code to keep your light responsive to the DMX data.

3.2 Troubleshooting

For proper operation, first ensure that both hardware and firmware are correctly configured by following the next validation steps. Once validated, you can begin reading DMX Channels and mapping them to the LEDs from your firmware.

Hardware Validation

First, ensure you have a proper hardware design by:

1. The VDD pin is supplied with 3.3 V.
2. Having an RS485 transceiver connected to RX/TX and DIR pins. The RX/TX pin must be pulled up with a 1 kΩ resistor to the transceiver supply voltage (either 3.3 V or 5 V).
3. Having your microcontroller connected via I2C on the SCL and SDA pins, with pull-up resistors to 3.3 V or 5 V.
4. Having the I2CSPD pin to VSS for 100 kHz, or any other configuration.

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

Firmware Validation

Before starting with your firmware, 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 IS3720 is detected on the bus.

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

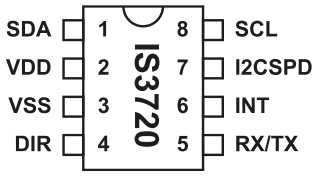
Once the IS3720 is detected on the bus, you can be confident that the I2C-serial interface is working correctly, and that any remaining issues will come from your firmware code.

At this point, you can implement your I2C read and write routine. The IS3720 uses **16-bit register addressing size with 8-bit data size**. Understanding this is crucial for proper implementation of read and write routines.

To verify your read routine, read the `CHIP_ID` register at address `513`. This register is read-only and always returns the fixed value `152`. If you obtain this value, communication is validated and you can safely begin your firmware.

Refer to Firmware Examples chapter for example codes in STM32, Arduino and Raspberry Pi platforms.

4 Pins



Pin	Name	Type	Description
1	SDA	Open-Drain 5 V-Tolerant	I2C data pin. Connects to your microcontroller. Open-drain pin; a pull-up resistor is required.
2	VDD	Supply	3.3 V power supply pin. Bypass this pin to VSS with a 100 nF and 2.2 µF ceramic capacitors. Place them as close as possible to the IC.
3	VSS	Supply	0 V reference pin.
4	DIR	Output Open-Drain 5 V-Tolerant	DMX+RDM UART Direction Pin. Open-drain pin; a pull-up resistor is required. Connect this pin to the Driver Enable (DE) and Receiver Enable (RE) pins of the RS485 transceiver, with a pull-up resistor connected to the transceiver supply voltage.
5	RX/TX	Input-Output Open-Drain 5 V-Tolerant	DMX+RDM UART receive and transmit pin at TTL voltage levels. Connect this pin to the Receiver data Output (RO) and Driver data Input (DI) pins of the RS485 transceiver, with a pull-up resistor connected to the transceiver supply voltage.
6	INT	Output Push-Pull	Interruption pin. This pin can optionally be connected to your microcontroller to notify changes to `DMX_START_ADDRESS` or `IDENTIFY_DEVICE` registers.
7	I2CSPD	Analog Input 0 to 3.3 V	I2C-Serial Interface Speed Selection pin. • For 100 kHz pull to VSS. • For 400 kHz make a voltage divider of VDD/2 (1.65 V). • For 1 MHz pull to VDD (3.3 V).
8	SCL	Open-Drain 5 V-Tolerant	I2C clock pin. Connects to your microcontroller. Open-drain; requires a pull-up resistor.

4.1 RX/TX and DIR Pins

DMX+RDM UART Communication Pins.

DMX+RDM communication is half-duplex; therefore, devices connected to this interface can operate either in receive or transmit mode, but not both simultaneously.

Since the communication is half-duplex, a single RX/TX pin is used for both reception and transmission, along with a data direction (DIR) pin.

An RS485 transceiver (e.g., THVD1500) is required to convert the logic levels of RX/TX pin to DMX+RDM (RS485) differential signals.

Both RX/TX and DIR pins are open-drain and require a pull-up resistor to the same supply voltage that the transceiver is working, typically 3.3 V or 5 V.

The RX/TX pin is connected to both the Driver data Input (DI) and the Receiver data Output (RO) of the RS485 transceiver.

The DIR pin controls the data direction. It is set high when the TX/RX pin is transmitting and low when receiving. This pin is connected to both the DE (Driver Enable) and RE (Receiver Enable) pins of the RS485 transceiver.

Using 5 V transceivers is preferred, as they provide better noise immunity and support longer cable lengths.

Refer to the Hardware Examples section for more details.

4.2 INT Pin

Interruption Pin.

This pin generates a positive pulse whenever the `DMX_START_ADDRESS` (70 ms) or `IDENTIFY_DEVICE` (50 ms) registers are modified.

Since these registers need be monitored for changes, this pin allows the microcontroller to avoid continuously polling them via I2C and instead relay on the pin status.

4.3 SCL and SDA Pins

I2C-Serial Interface Pins.

SCL (Serial Clock Line): This pin is used to synchronize data transfer between the IS3720 device and your microcontroller or other CPU.

SDA (Serial Data Line): This bidirectional pin is used for both sending and receiving data between the IS3720 and your microcontroller..

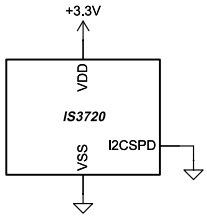
Both pins are open-drain and must be pulled up to 3.3 V or 5 V. The pull-up resistor value should be chosen based on the bus speed and capacitance. Typical values are 4.7 kΩ for Standard Mode (100 kHz) and 2 kΩ for Fast Mode and Fast Mode Plus (400 kHz and 1 MHz).

4.4 I2CSPD Pin

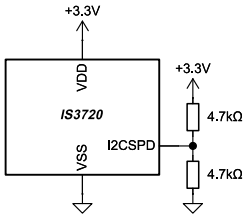
I2C-Serial Interface Speed Selection Pin.

This pin configures the IS3720 internal I2C-serial Interface timings and filters to properly work with the selected bus speed.

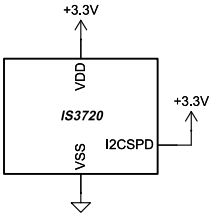
For a **100 kHz** setting, set the I2CSPD pin to VSS.



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 **1 MHz** setting, set the I2CSPD pin to 3.3 V.



5 Memory Map

The IS3720 memory map registers are 8 bit wide, and I2C memory addressing requires 2-bytes.

	I2C Register Address	Register Name	Register Description
DMX Registers	0	-	Not used.
	1	DMX_CH1	DMX Channel 1 value.
	2	DMX_CH2	DMX Channel 2 value.
	3	DMX_CH3	DMX Channel 3 value.
	...	...	DMX Channel 4 to Channel 509 values.
	510	DMX_CH510	DMX Channel 510 value.
	511	DMX_CH511	DMX Channel 511 value.
	512	DMX_CH512	DMX Channel 512 value.
	513	CHIP_ID	IS3720 chip identification number (does not change).
	514	CHIP_REV	IS3720 chip revision number (may change).
	515	CHIP_TEMP	IS3720 chip temperature in ºC (signed value).
RDM Registers	516 517	DMX_START_ADDRESS	DMX start address of your product.
	518	IDENTIFY_DEVICE	Device identification. When set to 1, your product flashes to help lighting technicians locate it.
	519 520	DMX_FOOTPRINT	Number of DMX channels used by your product.
	521 ... 526	UID	Unique RDM identifier of your product.
	527 528	CATEGORY	RDM-coded number classifying your product. See Table 2: RDM Standardized Product Categories.
	529 ... 560	MANUFACTURER_LBL	32-character ASCII string containing your product company name.
	561 562	MODEL_NUM	2-byte value identifying your product model.
	563 ... 594	MODEL_LBL	32-character ASCII string containing your product name.
	595 ... 598	SOFT_NUM	4-byte value identifying your product software version.
	599 ... 630	SOFT_LBL	32-character ASCII string representing the firmware name or code of your product.
	631	RDM_ONLINE	Enables or disables RDM communication (does not affect DMX) of your product.

Table 1: IS3720's Memory Map

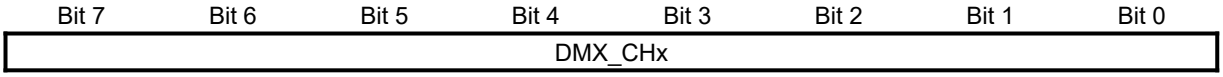
5.1 DMX_CHx

The DMX_CHx registers contain the values of their corresponding DMX channels received from the DMX controller. Each register number matches the DMX channel number. For example, the value of DMX Channel 12 is stored in the `DMX_CH12` register.

There are a total of 512 `DMX_CHx` registers, which are volatile RAM: if the chip loses power, the registers will reset to 0 on power-up until the next DMX frame is received. You can access and read any register individually or read a block of them.

These registers are read-only.

Name:	`DMX_CHx`
Description:	DMX Channel Values
Address Range:	`1` to `512`
Memory Type:	Volatile RAM
Allowed Values:	`0` to `255`
Reset Values:	`0`
Accessible by:	I2C (Only Read), DMX+RDM Controller (Only Write)



5.2 CHIP_ID

The `CHIP_ID` register contains the chip identifier, which is a fixed value of `152`. 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: Chip Identification Number
Address: `513`
Memory Type: ROM
Value: `152`
Accessible by: I2C (Read Only)

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

5.3 CHIP_REV

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`
Memory Type: ROM
Value: Depends on the revision
Accessible by: I2C (Read Only)

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

5.4 CHIP_TEMP

The `CHIP_TEMP` register indicates the internal IS3720 temperature.

This value is intended for internal chip calibration.

This register is read-only.

Name: `CHIP_TEMP`
Description: Internal Chip Temperature
Address: `515`
Memory Type: RAM
Value: Depends on the temperature, signed value (int8_t)
Accessible by: I2C (Read Only)

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

5.5 DMX_START_ADDRESS

The DMX_START_ADDRESS register contains the DMX start address of your device.

This register is very important in RDM, as it is one of the most frequently used parameters in the protocol.

Being able to change the DMX start address of a device remotely allows your product line to be easily deployed in large installations, such as building facades, or in locations where access is limited, such as waterproof fixtures or in-ground lights where opening the housing is not possible.

For products that are easily accessible, a DIP switch may also be present. In this case, a dedicated switch or a software menu may allow selecting whether the active DMX start address comes from the RDM register or from the DIP switch. Therefore, it is up to your firmware to decide in each situation which source will be the operative DMX start address.

It is the responsibility of your microcontroller to monitor this register block for any change on the DMX start address and read the new DMX channels accordingly.

To avoid continuous polling of this register block via I2C, a 70 ms signal will be generated on the INT pin whenever the content of this register block or the IDENTIFY_DEVICE register changes.

This is a 16-bit register composed by two 8-bit registers. Register address 516 is the high byte and register 517 is the low byte.

Allowed values for this register block are only DMX valid start addresses (1 to 512).

This register block can be read and written via I2C, and it can be also read and written via RDM.

This is a non-volatile register block. On each power-up, it loads its last stored value. Writing the same content to non-volatile memory that it previously held does not trigger a write cycle; therefore, it is safe to write to this register at every startup of your firmware.

Name:	DMX_START_ADDRESS
Description:	DMX Start Address
Address Range:	516 and 517
Memory Type:	Non-Volatile
Allowed Values:	1 to 512
Accessible by:	I2C (Read/Write), RDM (Read/Write)
RDM PID:	0x00F0 DMX_START_ADDRESS

Addresses:	Details:	Default Value:
516	High Byte	0x00
517	Low Byte	0x01

5.6 IDENTIFY_DEVICE

Identify your DMX device among others.

This is an important RDM function, typically used during lighting installation and configuration.

If the lighting operator is unsure which device corresponds to a specific UID, they can write a value of `1` to this register via RDM to make the device identify itself.

For example, imagine a scenario where a building facade has 50 lights, and the operator needs to identify each UID for proper DMX channel addressing later. An easy option is to activate the IDENTIFY_DEVICE functionality for the desired light UID. Without needing to set any DMX address, the light will identify itself by displaying an eye-catching pattern, allowing it to be easily distinguished from the rest.

For example, if it is a light, it could flash; if it is a smoke machine, it could release smoke, etc. Once the technician has located the device, they should set this register back to `0`.

The identification pattern is up to you. Your device must continue indicating itself for as long as this register remains set to `1`.

It is the responsibility of your microcontroller to monitor this register. When it is set to `1`, the normal DMX operation should be overridden and the identification sequence should be executed. Once the register is read back as `0`, normal DMX operation shall resume.

To avoid continuous polling of this register via I2C, a 50 ms signal will be generated on the INT pin whenever the content of this register or the DMX_START_ADDRESS changes.

This register corresponds to the RDM parameter 0x1000 IDENTIFY_DEVICE.

This is an 8-bit register and its allowed values are `0` or `1`.

This register block can be read and written via I2C and RDM.

This is a RAM register. On each power-up, it is set to `0`.

Name:	`IDENTIFY_DEVICE`
Description:	Identify Device
Address:	`518`
Memory Type:	RAM
Default Value:	`0`
Allowed Values:	`0` or `1`
Accessible by:	I2C (Read/Write), RDM (Read/Write)
RDM PID:	0x1000 IDENTIFY_DEVICE

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

5.7 DMX_FOOTPRINT

The DMX_FOOTPRINT register contains the quantity of DMX channels your device uses.

This register is very important in RDM, as it enables lighting operators to automatically patch your device. It tells the RDM controller how many DMX channels your device consumes. Using this information together with the DMX_START_ADDRESS, the lighting console can automatically patch your device at its location, reserve the required DMX slots, and assign the next RDM device to the next available DMX address — making a task that would normally take a long time fully automatic in just a second.

For example, let's assume your device is an RGB light. In that case, you will need three channels: one for red, one for green, and one for blue. Therefore, the DMX footprint of your fixture will be 3.

But color channels are not the only ones that consume DMX addresses. Let's suppose, in the previous example, that you use a fourth channel for the strobe frequency. In this case, the footprint of your device will be 4 — red, green, blue, and strobe frequency.

This register corresponds to the RDM parameter 0x0060 DEVICE_INFO.

This is a 16-bit register composed by two 8-bit registers. Register address 519 is the high byte of DMX_FOOTPRINT and register address 520 is the low byte.

Allowed values for this register block are 0 to 512.

This register block can be read and written via I2C, and it can be read via RDM.

This is a non-volatile register block. On each power-up, it loads its last stored value. Writing the same content to non-volatile memory that it previously held does not trigger a write cycle; therefore, it is safe to write to this register at every startup of your firmware.

Name:	DMX_FOOTPRINT
Description:	DMX Footprint
Address Range:	519 and 520
Memory Type:	Non-Volatile
Allowed Values:	0 to 512
Accessible by:	I2C (Read/Write), RDM (Read Only)
RDM PID:	0x0060 DEVICE_INFO.CURRENT_FOOTPRINT

Addresses:	Details:	Default Value:
519	High Byte	0x00
520	Low Byte	0x03

5.8 UID

The UID registers block contain the UID number, which is a unique RDM identifier of your device.

The UID is a block of 6 bytes. The first 2 bytes are the Manufacturer ID code, and the last 4 bytes is the product serial number.

The Manufacturer ID codes are assigned by the ESTA, you should apply for a Manufacturer ID code at ESTA. For the application site, please refer to the following link:

https://inacks.com/obtaining-rdm-id

Manufacturer IDs 0x7FF0 through 0x7FFF are reserved for prototyping and experimental use while a manufacturer is waiting for an assigned Manufacturer ID. They are not permitted in commercial products.

By default, the IS3720 uses the manufacturer code 0x7FF7, which falls within this reserved range. You should replace this code with your assigned Manufacturer ID.

By default, the IS3720 comes with a random serial number intended for prototyping and experimental use. This serial number may eventually be repeated. To ensure unique identification of your products, you should program your own serial numbers.

RDM messages send by the RDM controller can have 3 different destinations:

1. Unicast: The RDM message is for a specific device.
2. Full Broadcast: The RDM message is for all devices of the DMX+RDM network. This is achieved by using the special manufacturer code 0xFFFF and the special serial number 0xFFFFFFFF.
3. Manufacturer Broadcast: The RDM message is for all Responders of the same manufacturer. This is achieved by using the Manufacturer Code and the special Serial Number 0xFFFFFFFF.

This register block can be read and written via I2C, and read via RDM. Under a special provisioning condition, it can also be written via RDM; refer to the Provisioning section for more information.

An I2C write operation is only allowed when RDM is offline (RDM_ONLINE = 0). This protection prevents accidental UID changes during device operation.

In case of writing an invalid UID, the last valid UID will be used.

This is a non-volatile register block. On each power-up, it loads its last stored value. Writing the same content to non-volatile memory that it previously held does not trigger a write cycle; therefore, it is safe to write to this register at every startup of your firmware if need.

Name: UID
Description: Unique RDM Identifier Number of your device
Address Range: 521 to 526
Memory Type: Non-Volatile
Accessible by: I2C (Read/Write[2]), RDM (Read/Write[3])
RDM PID:

Addresses:	Details:		Default Value:	Valid Range:
521	UID, User ID	High Byte	0x7F	For production: Ask ESTA for an ID. For prototyping: 0x7FF0 to 0x7FFF.
521		Low Byte	0xF7	
522	UID, Serial Number	High Byte	Fixed Random Value	0x00000000 to 0xFFFFFFFE
523				
524				
526		Low Byte		

2 An I2C write operation to the UID is only allowed when RDM is offline (RDM_ONLINE = 0).
3 An RDM write operation is only allowed during Provisioning.

5.9 CATEGORY

The CATEGORY register block numerically identifies your device type according to the Table 2: RDM Standardized Product Categories. This values are common for all RDM devices.

This value appears on the lighting operator's console, helping to identify and operate the liginng system. It is for informational purposes only and does not affect RDM or DMX device operation.

The CATEGORY number is a 16-bit register block composed by two 8-bit registers. Register address 527 is the high byte and register address 528 is the low byte.

This register corresponds to the RDM parameter 0x0060 DEVICE_INFO.PRODUCT_CATEGORY.

This register block can be read and written via I2C and read via RDM.

This is a non-volatile register block. On each power-up, it loads its last stored value. Writing the same content to non-volatile memory that it previously held does not trigger a write cycle; therefore, it is safe to write to this register at every startup of your firmware.

This is a non-volatile register block. On each power-up, it loads its last stored value. Writing the same content to non-volatile memory that it previously held does not trigger a write cycle; therefore, it is safe to write to this register at every startup of your firmware.

Name: CATEGORY
Description: 2 bytes number to identify your product category
Address Range: 527 and 528
Memory Type: Non-Volatile
Allowed Values: Only values defined in the Table 2: RDM Standardized Product Categories
Accessible by: I2C (Read/Write), RDM (Read)
RDM PID: 0x0060 DEVICE_INFO.PRODUCT_CATEGORY

Addresses:		Default Value
527	High Byte	0x01
528	Low Byte	0x01

Table 2: RDM Standardized Product Categories

Product Category Defines	Register		Comment	
	527	528		
PRODUCT_CATEGORY_NOT_DECLARED	0x00	0x00	-	
Fixtures – intended as source of illumination				A fixture in this context is defined as a light source intended as a means of illumination. A projector is a light source capable of producing realistic images from another media. A light source intended for use with Fiber Optics should be classified as a fixture.
PRODUCT_CATEGORY_FIXTURE	0x01	0x00	No Fine Category declared	
PRODUCT_CATEGORY_FIXTURE_FIXED	0x01	0x01	No pan / tilt / focus style functions	
PRODUCT_CATEGORY_FIXTURE_MOVING_YOKE	0x01	0x02	-	
PRODUCT_CATEGORY_FIXTURE_MOVING_MIRROR	0x01	0x03	-	
PRODUCT_CATEGORY_FIXTURE_OTHER	0x01	0xFF	For example, focus but no pan/tilt.	
Fixture Accessories – add-ons to fixtures or projectors				
PRODUCT_CATEGORY_FIXTURE_ACCESSORY	0x02	0x00	No Fine Category declared	
PRODUCT_CATEGORY_FIXTURE_ACCESSORY_COLOR	0x02	0x01	Scrollers / Color Changers	
PRODUCT_CATEGORY_FIXTURE_ACCESSORY_YOKE	0x02	0x02	Yoke add-on	
PRODUCT_CATEGORY_FIXTURE_ACCESSORY_MIRROR	0x02	0x03	Moving mirror add-on	
PRODUCT_CATEGORY_FIXTURE_ACCESSORY_EFFECT	0x02	0x04	Effects Discs	
PRODUCT_CATEGORY_FIXTURE_ACCESSORY_BEAM	0x02	0x05	Gobo Rotators / Iris / Shutters / Dousers Beam modifiers	
PRODUCT_CATEGORY_FIXTURE_ACCESSORY_OTHER	0x02	0xFF	-	
Projectors – light source capable of producing realistic images from another media			Video / Slide / Oil Wheel / Film / LCD	
PRODUCT_CATEGORY_PROJECTOR	0x03	0x00	No Fine Category declared	
PRODUCT_CATEGORY_PROJECTOR_FIXED	0x03	0x01	No pan / tilt functions	
PRODUCT_CATEGORY_PROJECTOR_MOVING_YOKE	0x03	0x02	-	
PRODUCT_CATEGORY_PROJECTOR_MOVING_MIRROR	0x03	0x03	-	
PRODUCT_CATEGORY_PROJECTOR_OTHER	0x03	0xFF	-	

Product Category Defines	Register		Comment
	527	528	
Atmospheric Effect – earth/wind/fire			
PRODUCT_CATEGORY_ATMOSPHERIC	0x04	0x00	No Fine Category declared
PRODUCT_CATEGORY_ATMOSPHERIC_EFFECT	0x04	0x01	Fogger / Hazer / Flame, etc.
PRODUCT_CATEGORY_ATMOSPHERIC_PYRO	0x04	0x02	The DMX512 standard specifically states that, as there is no mandatory error checking, DMX512 is not an appropriate control protocol for hazardous applications. RDM may, however, be appropriate for configuration and monitoring purposes.
PRODUCT_CATEGORY_ATMOSPHERIC_OTHER	0x04	0xFF	-
Intensity Control (specifically Dimming equipment)			
PRODUCT_CATEGORY_DIMMER	0x05	0x00	No Fine Category declared
PRODUCT_CATEGORY_DIMMER_AC_INCANDESCENT	0x05	0x01	AC > 50VAC
PRODUCT_CATEGORY_DIMMER_AC_FLUORESCENT	0x05	0x02	-
PRODUCT_CATEGORY_DIMMER_AC_COLDCATHODE	0x05	0x03	High Voltage outputs such as Neon or other cold cathode.
PRODUCT_CATEGORY_DIMMER_AC_NONDIM	0x05	0x04	Non-Dim module in dimmer rack
PRODUCT_CATEGORY_DIMMER_AC_ELV	0x05	0x05	AC <= 50V such as 12/24V AC Low voltage lamps.
PRODUCT_CATEGORY_DIMMER_AC_OTHER	0x05	0x06	-
PRODUCT_CATEGORY_DIMMER_DC_LEVEL	0x05	0x07	Variable DC level output
PRODUCT_CATEGORY_DIMMER_DC_PWM	0x05	0x08	Chopped (PWM) output.
PRODUCT_CATEGORY_DIMMER_CS_LED	0x05	0x09	Specialized LED dimmer
PRODUCT_CATEGORY_DIMMER_OTHER	0x05	0xFF	-
Power Control (other than Dimming equipment)			
PRODUCT_CATEGORY_POWER	0x06	0x00	No Fine Category declared
PRODUCT_CATEGORY_POWER_CONTROL	0x06	0x01	Contactor racks, other forms of Power Controllers
PRODUCT_CATEGORY_POWER_SOURCE	0x06	0x02	Generators
PRODUCT_CATEGORY_POWER_OTHER	0x06	0xFF	-
Scenic Drive – including motorized effects unrelated to light source			
PRODUCT_CATEGORY_SCENIC	0x07	0x00	-
PRODUCT_CATEGORY_SCENIC_DRIVE	0x07	0x01	-
PRODUCT_CATEGORY_SCENIC_OTHER	0x07	0xFF	-
DMX Infrastructure, conversion and interfaces			
PRODUCT_CATEGORY_DATA	0x08	0x00	No Fine Category declared
PRODUCT_CATEGORY_DATA_DISTRIBUTION	0x08	0x01	Splitters / repeaters / data patch / Ethernet products used to distribute DMX universes.
PRODUCT_CATEGORY_DATA_CONVERSION	0x08	0x02	Protocol Conversion analog decoders.
PRODUCT_CATEGORY_DATA_OTHER	0x08	0xFF	-
Audio-Visual Equipment			
PRODUCT_CATEGORY_AV	0x09	0x00	-
PRODUCT_CATEGORY_AV_AUDIO	0x09	0x01	No Fine Category declared
PRODUCT_CATEGORY_AV_VIDEO	0x09	0x02	Audio controller or device
PRODUCT_CATEGORY_AV_OTHER	0x09	0xFF	Video controller or display device
Parameter Monitoring Equipment			This category is for equipment that has no DMX512 control capability, but uses RDM to provide a data logging or monitoring function.
PRODUCT_CATEGORY_MONITOR	0x0A	0x00	No Fine Category declared
PRODUCT_CATEGORY_MONITOR_ACLINEPOWER	0x0A	0x01	Product that monitors AC line voltage, current or power
PRODUCT_CATEGORY_MONITOR_DCPOWER	0x0A	0x02	Product that monitors DC line voltage, current or power
PRODUCT_CATEGORY_MONITOR_ENVIRONMENTAL	0x0A	0x03	Temperature or other environmental parameter
PRODUCT_CATEGORY_MONITOR_OTHER	0x0A	0xFF	-
Controllers, Backup devices			
PRODUCT_CATEGORY_CONTROL	0x70	0x00	No Fine Category declared
PRODUCT_CATEGORY_CONTROL_CONTROLLER	0x70	0x01	-
PRODUCT_CATEGORY_CONTROL_BACKUPDEVICE	0x70	0x02	-
PRODUCT_CATEGORY_CONTROL_OTHER	0x70	0xFF	-
Test Equipment			
PRODUCT_CATEGORY_TEST	0x71	0x00	No Fine Category declared
PRODUCT_CATEGORY_TEST_EQUIPMENT	0x71	0x01	-
PRODUCT_CATEGORY_TEST_EQUIPMENT_OTHER	0x71	0xFF	-
Miscellaneous			
PRODUCT_CATEGORY_OTHER	0x7F	0xFF	For devices that aren't described within this table

5.10 MANUFACTURER_LBL

The MANUFACTURER_LBL register block contains a text with the device manufacturer's name. It must be consistent between all products manufactured within an ESTA Manufacturer ID, refer to section UID for more details.

This text appears on the lighting operator's console, it is for informational purposes only and does not affect RDM or DMX device operation.

This register is filled with text string and corresponds to the RDM parameter 0x0081 MANUFACTURER_LABEL. This register composed by a block of 32 8-bit registers.

Register address 529 is the high byte (first character) and register address 560 is the low byte (last character).

Allowed values for this register are any readable ASCII characters. The null character (value 0) is also valid and may be used for string termination.

This register block can be read and written via I2C, and read via RDM.

This is a non-volatile register block. On each power-up, it loads its last stored value. Writing the same content to non-volatile memory that it previously held does not trigger a write cycle; therefore, it is safe to write to this register at every startup of your firmware.

Name:	MANUFACTURER_LBL
Description:	32 characters string with the company name
Address Range:	529 to 560
Memory Type:	Non-Volatile
Allowed Values:	Readable ASCII characters and null character
Accessible by:	I2C (Read/Write), RDM (Read Only)
RDM PID:	0x0081 MANUFACTURER_LABEL

Addresses:	Details:	Default Chars:
529	High Byte (First Character)	'I'
530		'N'
531		'A'
532		'C'
533		'K'
534		'S'
535		'\0'
536		'\0'
537		'\0'
538		'\0'
539		'\0'
540		'\0'
541		'\0'
542		'\0'
543		'\0'
544		'\0'
545		'\0'
546		'\0'
547		'\0'
548		'\0'
549		'\0'
550		'\0'
551		'\0'
552		'\0'
553		'\0'
554		'\0'
...		...
558		'\0'
559		'\0'
560	Low Byte (Last Character)	'\0'

5.11 MODEL_NUM

The MODEL_NUM register block numerically identifies your device model.

This number appears on the lighting operator's console, it is for informational purposes only and does not affect RDM or DMX device operation.

The MODEL_NUM number is a 16-bit register block composed by two 8-bit registers. Register address 561 is the high byte and register address 562 is the low byte.

This register corresponds to the RDM parameter 0x0060 DEVICE_INFO.DEVICE_MODEL_ID.

This register block can be read and written via I2C and read via RDM.

This is a non-volatile register block. On each power-up, it loads its last stored value. Writing the same content to non-volatile memory that it previously held does not trigger a write cycle; therefore, it is safe to write to this register at every startup of your firmware.

Name:	MODEL_NUM
Description:	2 bytes number to identify your light model
Address Range:	561 and 562
Memory Type:	Non-Volatile
Allowed Values:	0x0000 to 0xFFFF
Accessible by:	I2C (Read/Write), RDM (Read)
RDM PID:	0x0060 DEVICE_INFO.DEVICE_MODEL_ID

Addresses:	Details:	Default Value:
561	High Byte	0x00
562	Low Byte	0x01

5.12 MODEL_LBL

The MODEL_LBL register block contains a text description for the device model number (MODEL_NUM).

This text appears on the lighting operator's console, it is for informational purposes only and does not affect RDM or DMX device operation.

This register is filled with text string and corresponds to the RDM parameter 0x0080 DEVICE_MODEL_DESCRIPTION.

This register composed by a block of 32 8-bit registers. Register address 563 is the high byte (first character) and register address 594 is the low byte (last character).

Allowed values for this register are any readable ASCII characters. The null character (value 0) is also valid and may be used for string termination.

This register block can be read and written via I2C, and read via RDM.

This is a non-volatile register block. On each power-up, it loads its last stored value. Writing the same content to non-volatile memory that it previously held does not trigger a write cycle; therefore, it is safe to write to this register at every startup of your firmware.

Name:	MODEL_LBL
Description:	32 characters string with the device model name
Address Range:	563 to 594
Memory Type:	Non-Volatile
Allowed Values:	Readable ASCII characters and null character
Accessible by:	I2C (Read/Write), RDM (Read)
RDM PID:	0x0080 DEVICE_MODEL_DESCRIPTION

Addresses:	Details:	Default Char:
563	High Byte (First Character)	'I'
564		'S'
565		'3'
566		'7'
567		'2'
568		'0'
569		' '
570		'I'
571		'2'
572		'C'
573		' '
574		'D'
575		'M'
576		'X'
577		'+'
578		'R'
579		'D'
580		'M'
581		' '
582		'R'
583		'e'
584		'c'
585		'e'
586		'i'
587		'v'
588		'e'
589		'r'
590		' '
591		'I'
592		'C'
593		'\0'
594	Low Byte (Last Character)	'\0'

5.13 SOFT_NUM

The SOFT_NUM register block numerically indicates the software version number of your device.

This number appears on the lighting operator's console, it is for informational purposes only and does not affect RDM or DMX device operation.

All devices from the same manufacturer and with the same MODEL_NUM, which use identical software, shall report back the same SOFT_NUM, and shall use identical software.

The SOFT_NUM number is a 16-bit register block composed by four 8-bit registers. Register address 595 is the high byte and register address 598 is the low byte.

This register corresponds to the RDM parameter 0x0060 DEVICE_INFO.SOFTWARE_VERSION_ID.

This register block can be read and written via I2C and read via RDM.

This is a non-volatile register block. On each power-up, it loads its last stored value. Writing the same content to non-volatile memory that it previously held does not trigger a write cycle; therefore, it is safe to write to this register at every startup of your firmware.

Name:	SOFT_NUM
Description:	4 bytes number to identify the software number
Address Range:	595 and 598
Memory Type:	Non-Volatile
Allowed Values:	0x0000 to 0xFFFF
Accessible by:	I2C (Read/Write), RDM (Read)
RDM PID:	0x0060 DEVICE_INFO.SOFTWARE_VERSION_ID

Addresses:	Details:	Default Value:
595	High Byte	0x00
596		0x01
597		0x00
598	Low Byte	0x00

5.14 SOFT_LBL

The SOFT_LBL register block provides a descriptive text label for the device's software version number (SOFT_NUM).

This text appears on the lighting operator's console, it is for informational purposes only and does not affect RDM or DMX device operation.

This register corresponds to the RDM parameter 0x00C0 SOFTWARE_VERSION_LABEL.

This register composed of a block of 32 8-bit registers. Register address 599 contains the first character and register address 630 contains the last character.

Allowed values for this register are any readable ASCII characters. The null character (value 0) is also valid and may be used for string termination.

This register block can be read and written via I2C, and read via RDM.

This is a non-volatile register block. On each power-up, it loads its last stored value. Writing the same content to non-volatile memory that it previously held does not trigger a write cycle; therefore, it is safe to write to this register at every startup of your firmware.

Name:	SOFT_LBL
Description:	32 characters string with the model name
Address Range:	599 to 630
Memory Type:	Non-Volatile
Allowed Values:	Readable ASCII characters and null character
Accessible by:	I2C (Read/Write), RDM (Read)
RDM PID:	0x00C0 SOFTWARE_VERSION_LABEL

Addresses:	Details:	Default Char:
599	High Byte (First Character)	'V'
600		'e'
601		'r'
602		's'
603		'i'
604		'o'
605		'n'
606		' '
607		'1'
608		'.'
609		'0'
610		'0'
611		'\0'
612		'\0'
613		'\0'
614		'\0'
615		'\0'
616		'\0'
617		'\0'
618		'\0'
619		'\0'
620		'\0'
621		'\0'
622		'\0'
623		'\0'
624		'\0'
625		'\0'
626		'\0'
627		'\0'
628		'\0'
629		'\0'
630	Low Byte (Last Character)	'\0'

5.15 RDM_ONLINE

The `RDM_ONLINE` register enables or disables the RDM functionality of the IS3720. DMX reception remains active regardless of the value of this register.

By default, the chip powers up with this register set to `0`, allowing time to configure the RDM parameters (the UID, manufacturer label, model label, etc.). Once the RDM parameters are configured, the chip can be brought online by writing a `1` to this register.

When this register is set to `1`, the `UID` register cannot be modified. This mechanism prevents accidental `UID` register modification while your product is operating.

Writing the key value `160` to this register enables a special PID (0x8000), called “Update UID”. Writing to this PID via RDM allows custom UID provisioning.

This register is not associated with any RDM PID. It is 8-bit register that accepts values `0` and `1`. It’s memory type is RAM, and it is reset to `0` at each power-up.

This is a RAM register. On each power-up, it is set to `0`.

Name:	`RDM_ONLINE`
Description:	Enables or disables the RDM functionality of the IC
Address:	`631`
Memory Type:	RAM
Default Value:	`0`
Allowed Values:	`0` or `1`
Accessible by:	Only I2C (Read/Write)

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

6 I2C Description

The IS3720 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 IS3720 is represented on the bus by the I2C device address: 22.

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 IS3720'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 632 registers; therefore, addressing a register requires 2 bytes. Each register is 8 bits wide.

The IS3720 requires clock stretching.

The operability of the Read and Write commands of the IS3720 is very similar to an EEPROM memory. Thinking of the IS3720 as an EEPROM memory is a good analogy to quickly understand how to communicate with the device.

6.1 Highlights

I2C Device Address: 22

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 632 registers)
- Single-Byte Read
- Multiple-Byte Read (up to 632 registers)

Overreading and Overwriting the memory:

- If a write operation starts at a valid memory address (`0` to `631`) 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 `631`) will result in a NACK and data will be discarded.
- If a read operation starts at a valid memory address (`0` to `631`) 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 `631`) will return a value of 0xFF.

6.2 Single Byte Write

Writing a single byte is an action performed by your microcontroller (I2C-master) to write data to any register within the IS3720 memory map (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 IS3720's pointer register. Once the address is set, your microcontroller can send the data to be written.

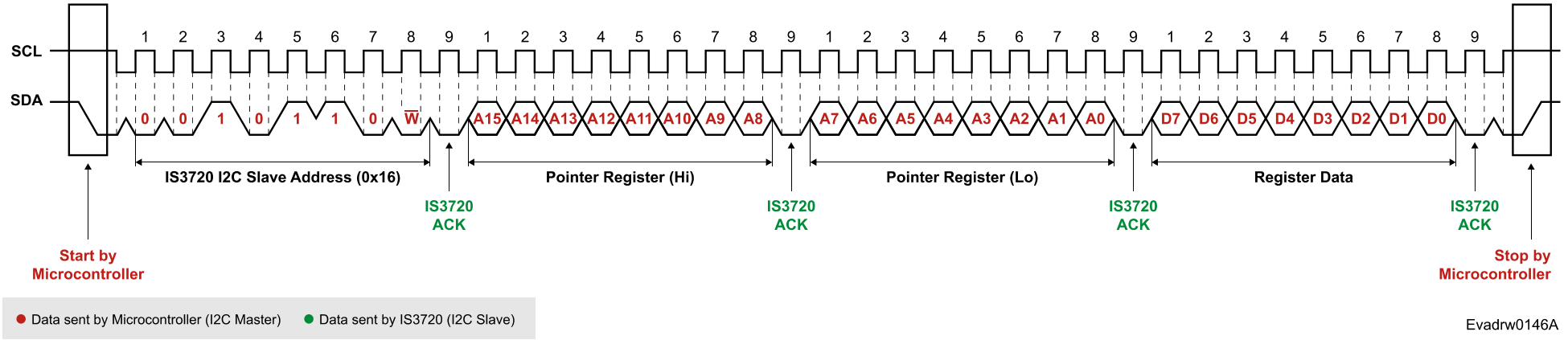
To initiate the single byte write operation, the following steps must be performed from the beginning: your microcontroller begins by pulling down the SDA line while the SCL line is high, creating a Start Condition. It then sends the IS3720 I2C device address 22 with the R/W bit set to `0` (indicating a write operation).

Upon receiving the device address, the IS3720 acknowledges it. Subsequently, your 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 IS3720.

Your microcontroller then sends the byte to be written, which the IS3720 acknowledges. Finally, your 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 IS3720 for a write operation goes from addresses `0` to `631`. If a Write Operation is performed with a Pointer Register higher than `631`, the IS3720 will answer with a NACK.



Evadrw0146A

6.3 Multiple Byte Write

The Multiple Byte Write operation functions similarly to the Single Byte Write, but allows writing a block of up to 632 registers in a single operation.

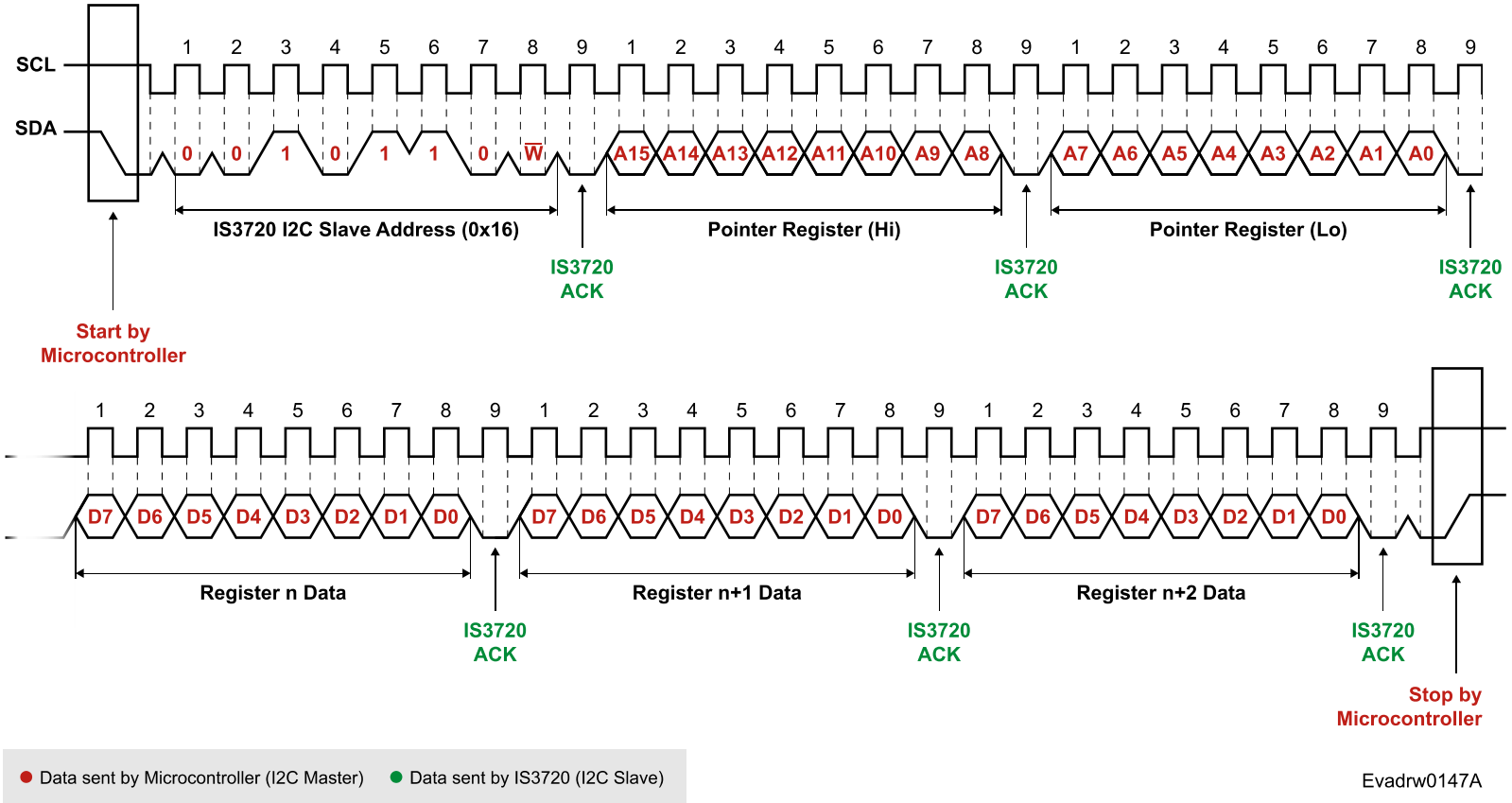
To perform a Multiple Byte Write operation, follow the same procedure as for a Single Byte Write until the first data byte is written. After writing the first byte, instead of generating a Stop Condition, your microcontroller should continue writing data bytes. To conclude the write operation, after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

Invalid Memory Addressing

The valid memory range of the IS3720 for a write operation goes from addresses `0` to `631`.

If a Multiple Byte Write Operation is performed with a pointer register within the valid memory range (`0` to `631`) but exceeds the last memory register (`631`), a rollover to register `0` will occur.

If a Multiple Byte Write Operation is performed with a pointer register outside from the valid memory range (greater than `631`), the IS3720 will respond with a NACK upon receiving the first data byte.



6.4 Single Byte Read

Reading a single byte is an action performed by your microcontroller (I2C-master) to access any register within the IS3720 memory map (I2C-slave), regardless of the last read position. To perform this action, your microcontroller must load the address of the register to be read into the IS3720's pointer register. Once the address is set, the microcontroller can retrieve the data from the specified register.

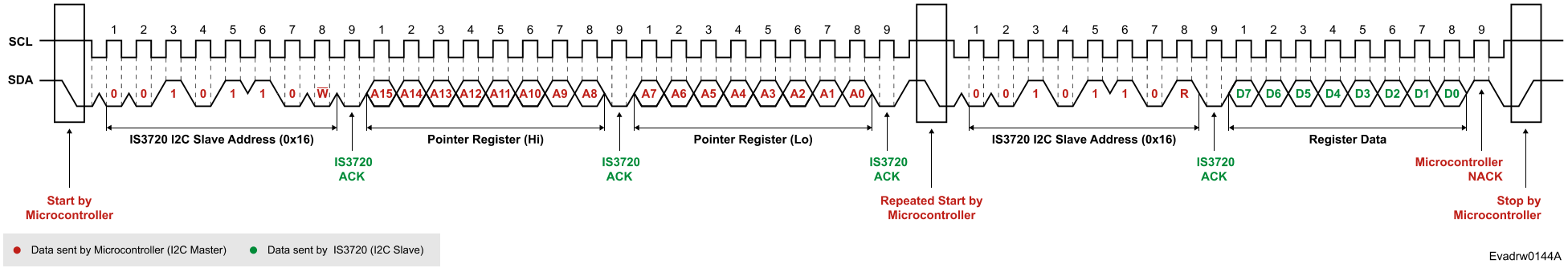
To initiate the Single Byte Read operation, the following steps must be performed from the beginning: your microcontroller starts by pulling SDA low while SCL is high to generate a Start Condition. It then sends the IS3720 I2C device slave address 22 with the R/W bit set to `0` (write). Upon receiving the device address, the IS3720 acknowledges it. Subsequently, your microcontroller sends the two bytes of the pointer register address: the most significant byte first, followed by the less significant byte, each acknowledged by the IS3720. This sets the address of the next register to be read in the pointer register.

Next, the content of the pointer register needs to be read.

Your microcontroller generates a Repeated Start Condition, followed by the IS3720 I2C device address 22 with the R/W bit set to `1` (read), instructing the IS3720 to retrieve data. The IS3720 acknowledges and responds with the data, which your 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 IS3720 goes from addresses `0` to `631`. If a Single Byte Read operation is performed with a pointer register higher than `631`, the read result will be 0xFF.



Evadrw0144A

6.5 Multiple Byte Read

Multiple Byte Read operation functions similarly to Single Byte Read but can read a block of up to 632 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), your microcontroller should continue acknowledging (ACK) each received data byte from the IS3720 for as many bytes as it intends to read. To conclude the read operation, after reading the last data byte, your microcontroller should generate a Not Acknowledge (NACK) and a Stop Condition.

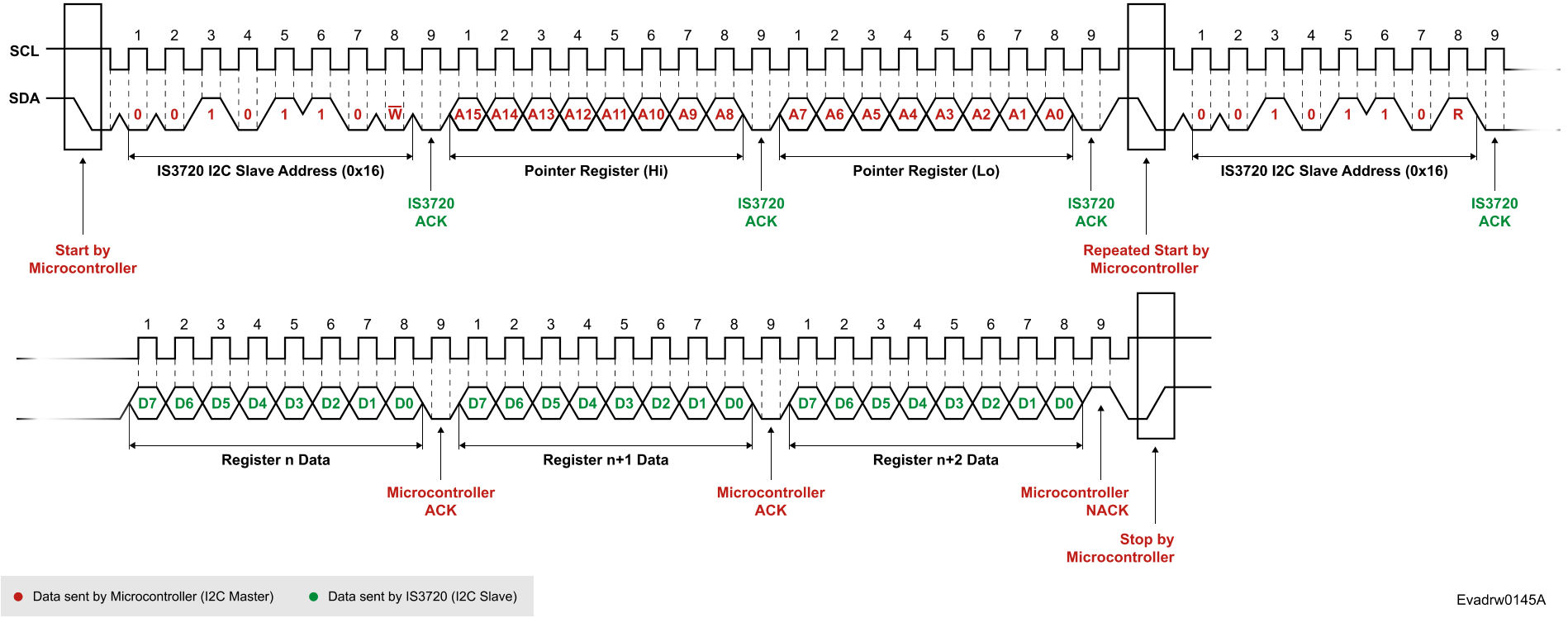
With each byte read, the pointer register increments by one.

Invalid Memory Addressing

The valid memory range of the IS3720 goes from addresses `0` to `631`.

If the Multiple Byte Read operation is performed with a pointer register within the valid memory range (`0` to `631`), but the data retrieval extends beyond register `631`, a rollover to position `0` will occur. For example, the value of register `633` will correspond to the content of register `1` (`DMX_CH1`).

If a Multiple Byte Read operation is performed with a pointer register value higher than `631`, the read result will be `0xFF`.



Evadrw0145A

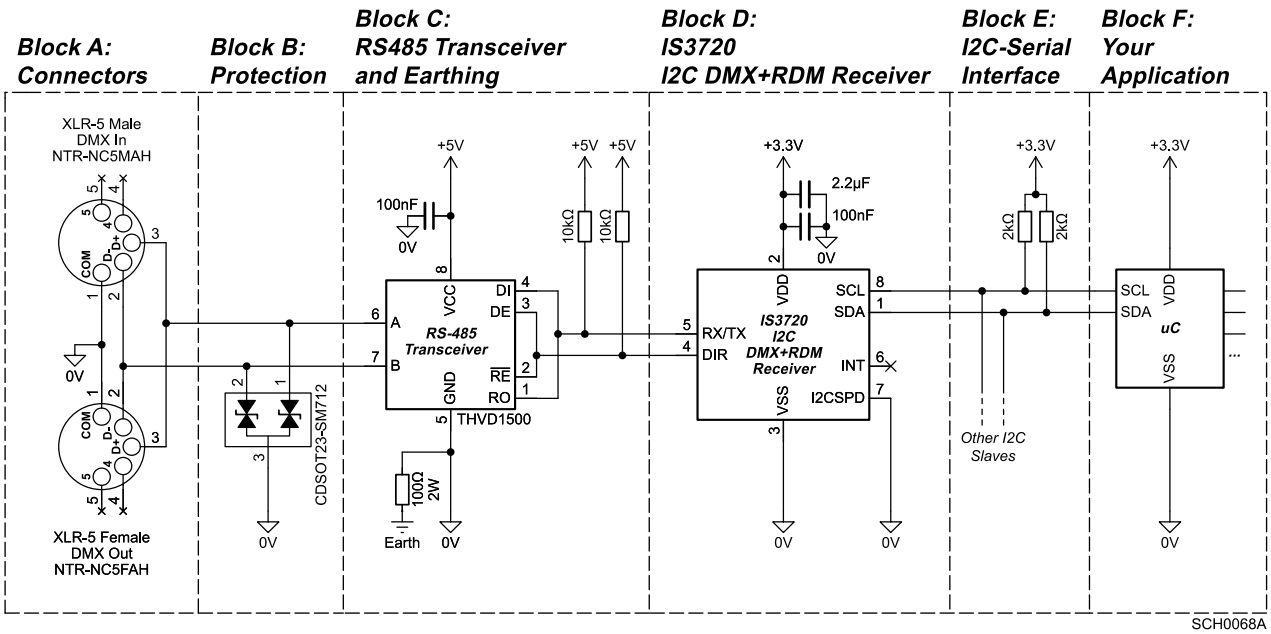
7 Hardware Examples

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 DMX and RDM specifications.

7.1 DMX+RDM Responder

This example shows the design of a DMX+RDM Receiver using a non-isolated receiver.

The standard requires the manufacturers to label this type of receiver as **NON-ISO**.



Block A: Connectors

Connectors

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.

XLR-5 connectors are typically used in professional equipment, while XLR-3 connectors are more common in cost-sensitive devices. XLR-3 is generally cheaper than XLR-5. 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.

DMX receivers (with or without RDM) use two XLR-5 connectors in daisy-chain configuration with the following configuration:

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

And the following pinout:

- Pin 1: Singal-Common
- Pin 2: Pair A, Data −
- Pin 3: Pair A, Data +
- Pin 4: Pair B, not used
- Pin 5: Pair B, not used

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.

Terminator

If your device is the last one on the DMX bus, a line terminator should be placed at its DMX Out connector. This reduces signal reflections and prevents DMX signal degradation, which can cause flickering on long cable runs among other problems.

The line terminator is a 120 Ω, 1 W resistor connected across the balanced signal lines (Data + and Data −). You can also find DMX connectors (with no cable) that have this resistor integrated internally.

Place only line terminators at the beginning and at the end of the DMX bus. No more than two terminators should be used on the same bus. Usually, DMX controllers include an internal terminator, so adding one at the beginning of the bus is often unnecessary.

Block B: Protection

The protection stage is influenced by several factors, including the intrinsic robustness and protection features of the RS485 transceiver 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 C: Transceiver

DMX operates over the RS485 electrical standard. Therefore, an RS485 transceiver is required to convert the differential RS485 signals to TTL-compatible voltage levels before entering the IS3720.

Either 3.3 V or 5 V transceivers can be used, as the IS3720 RX/TX and DIR pins are 5 V tolerant. However, 5 V devices are preferred because they offer better noise immunity on the DMX bus and allow longer cable runs.

Grounding-Earthing

Do not connect the pin 1 (Singal-Common) of the XLR-5 connector directly to the earth (the mains earth), as this can create dangerous current loops.

Signal-Common should be connected to earth through a 100 Ω (2 W) resistor to limit potential current flow through the DMX cable. This is especially important in large installations.

Block D: IS3720

The IS3720 is very simple to integrate into your design.

A 100 nF and 2.2 µF decoupling capacitors should be placed as close as possible to the supply pins (VDD and VSS).

The RX/TX is the UART data pin for DMX+RDM communication. This pin requires an external pull-up resistor. Since DMX+RDM communication is half-duplex, this pin connects to both the Driver Input and Receiver Output of the transceiver.

The DIR pin connects to both the Driver Enable and Receiver Enable pins. The DIR pin also requires a pull-up resistors. This pin goes high when the IS3720 is transmitting data and remains low at all other times.

Use the same pull-up voltage as the transceiver supply. If you use a 5 V transceiver, connect the pull-up resistors to 5 V. If you use a 3.3 V transceiver, connect the pull-up resistors to 3.3 V.

The I2CSPD pin defines the I2C speed. Connect this pin to VSS 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 VSS. For 1 MHz, the pin must be connected to 3.3 V. This pin is not 5 V tolerant.

Block E: I2C-Serial Interface

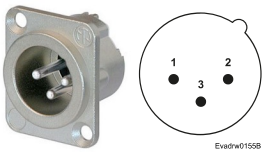
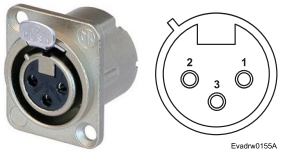
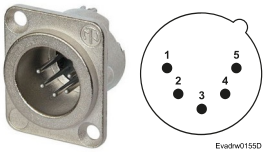
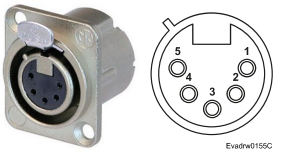
For proper operation of the I2C-serial interface, pull-up resistors to 3.3 V or 5 V are necessary. Typical resistor values are 4.7 kΩ for Standard Mode (100 kHz) and 2 kΩ for both Fast Mode (400 kHz) and Fast Mode Plus (1 MHz).

Block F: Application

Here is the rest of your product design. Typically, a microcontroller interfaces with the IS3720, 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.

Your task here is to read the register DMX_START_ADDRESS, then read the corresponding DMX_CHx channels data registers and map their values to your LEDs.

7.2 DMX+RDM Pinout

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

8 Firmware Examples

The following chapter provides example firmware for demonstration purposes only and does not form part of the product specification. The customer is responsible for implementing their own solution and for ensuring compliance with applicable legislation and the DMX and RDM specifications.

8.1 STM32

Coding the IS3720 on an STM32 is very simple. You do not need any special libraries—using the I2C HAL functions `HAL_I2C_Mem_Read()` and `HAL_I2C_Mem_Write()` is sufficient to communicate with the IS3720.

In the example, the first operation is to enable RDM communication by writing a `1` to the `RDM_ONLINE` register (address `631`). Once this is done, you only need to continuously check the `DMX_START_ADDRESS` register (address `516` and `517`) to ensure you always read the correct DMX channels assigned to your device. This is important because the lighting operator may change your DMX Start Address via RDM, which is one of the most commonly used RDM features.

Once you have read your DMX data, you can send it to your LEDs using PWM or whatever control method your product implements. The `pwm()` functions in the example code are provided for reference only.

Before enabling RDM, typical operations include setting the product information, such as the manufacturer label, software version, device category, etc.

In the `while(1)` loop, it is also necessary to regularly check the `IDENTIFY_DEVICE` register to determine whether the lighting operator has requested the device to identify itself.

```
// Enable RDM operation
uint8_t startRDM[1];
startRDM[0] = 1;
HAL_I2C_Mem_Write(&hi2c1, (22 << 1), 631, I2C_MEMADD_SIZE_16BIT, startRDM, 1, 1000);

while (1) {
	uint8_t readDmxStartAddress[2];
	uint16_t dmxStartAddress;
	uint8_t readDmxChannels[3];

	// Read the DMX start address assigned via RDM
	HAL_I2C_Mem_Read(&hi2c1, (22 << 1), 516, I2C_MEMADD_SIZE_16BIT, readDmxStartAddress, 2, 1000);
	dmxStartAddress = readDmxStartAddress[0] << 8;
	dmxStartAddress = dmxStartAddress | readDmxStartAddress[1];

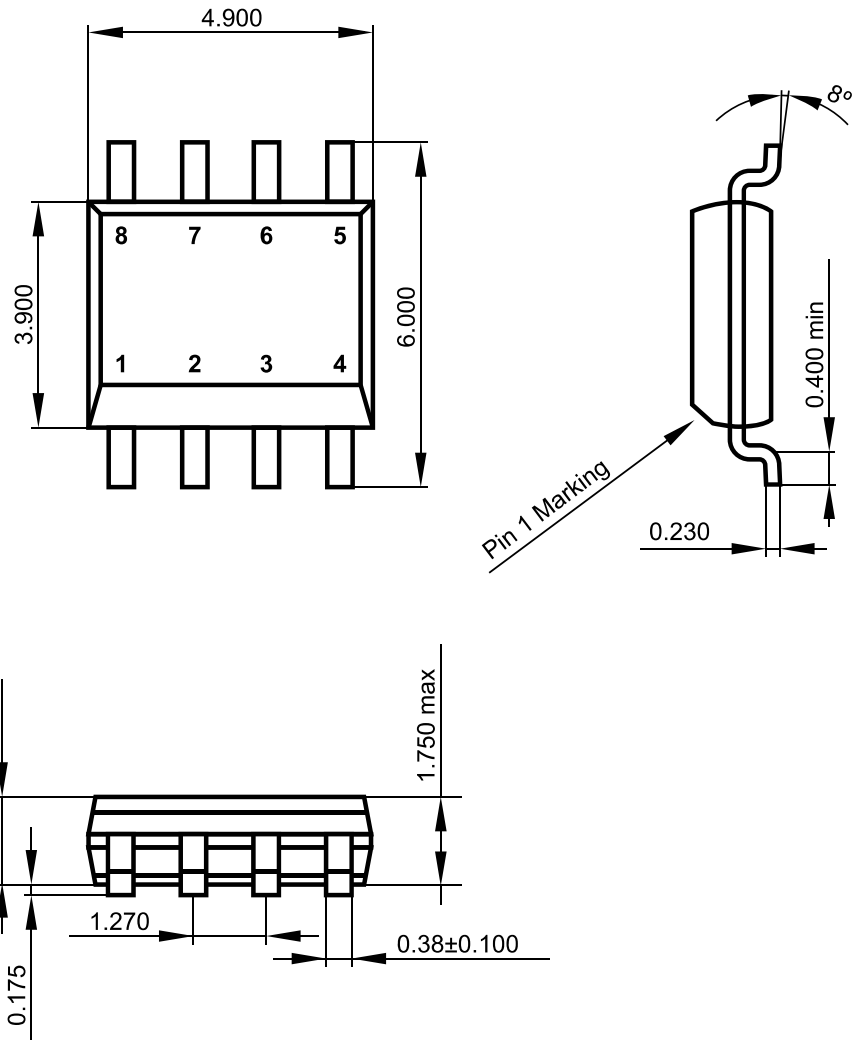
	// Read DMX channel data starting at the start address
	HAL_I2C_Mem_Read(&hi2c1, (22 << 1), dmxStartAddress, I2C_MEMADD_SIZE_16BIT, readDmxChannels, 3, 1000);

	// Update PWM outputs
	pwm(channelRed, readDmxChannels[0]);
	pwm(channelGreen, readDmxChannels[1]);
	pwm(channelBlue, readDmxChannels[2]);
}
```

ISFW1090

9 Mechanical

SO8N Package

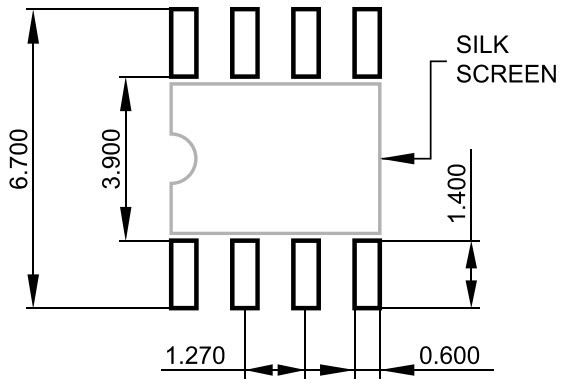


Units: millimeters

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

Evadrw0033B

SO8N Recommended Footprint



Units: millimeters

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

Evadrw0025r4

Ordering Information

Standard configuration

The standard configuration refers to the device variant that is mass-produced and offered at the most cost-efficient price.

Orderable Device	Status	Package		Pins	Temperature Range (Junction)
IS3720-S8I	ACTIVE		SO8N (4.9×6 mm)	8	-40ºC to 85ºC (105ºC)
IS3720-BBI	ACTIVE		Bread Board (10×14 mm)	8	-40ºC to 85ºC (105ºC)

Special configurations

Special configurations are produced upon request only and are subject to a minimum order quantity. Please contact sales (Sales Contact) for more information.

Orderable Device	Status	Package		Pins	Temperature Range (Junction)
IS3720-S8E	SPECIAL		SO8N (4.9×6 mm)	8	-40ºC to 105ºC (125ºC)
IS3720-S8H					-40ºC to 125ºC (130ºC)
IS3720-W12I			WLCSP (1.7×1.42 mm)	12	-40ºC to 85ºC (105ºC)
IS3720-W12E					-40ºC to 105ºC (125ºC)
IS3720-W12H					-40ºC to 125ºC (130ºC)
IS3720-U20I			UFQFPN (3×3 mm)	20	-40ºC to 85ºC (105ºC)
IS3720-U20E					-40ºC to 105ºC (125ºC)
IS3720-U20H					-40ºC to 125ºC (130ºC)

ACTIVE: Product recommended for new designs.

SPECIAL: Available only for high-volume pre-orders. Contact sales.

NRND: Not recommended for new designs. Device is in production to support existing customers, but it is not recommended for use in new designs.

OBSOLETE: Production of the device has been discontinued.

Content

IS3720: I2C DMX+RDM Receiver IC 1
1 Specifications 2
2 Description 4
2.1 IS3720 Description 4
2.2 Provisioning 4
3 Usage 5
3.1 Example: Designing an underwater RGB light with RDM 5
3.2 Troubleshooting 6
4 Pins 7
4.1 RX/TX and DIR Pins 7
4.2 INT Pin 7
4.3 SCL and SDA Pins 7
4.4 I2CSPD Pin 8
5 Memory Map 9
5.1 DMX_CHx 10
5.2 CHIP_ID 11
5.3 CHIP_REV 11
5.4 CHIP_TEMP 11
5.5 DMX_START_ADDRESS 12
5.6 IDENTIFY_DEVICE 13
5.7 DMX_FOOTPRINT 14
5.8 UID 15
5.9 CATEGORY 16
5.10 MANUFACTURER_LBL 19
5.11 MODEL_NUM 20
5.12 MODEL_LBL 21
5.13 SOFT_NUM 22
5.14 SOFT_LBL 23
5.15 RDM_ONLINE 24
6 I2C Description 25
6.1 Highlights 25
6.2 Single Byte Write 26
6.3 Multiple Byte Write 27
6.4 Single Byte Read 28
6.5 Multiple Byte Read 29
7 Hardware Examples 30
7.1 DMX+RDM Responder 30
7.2 DMX+RDM Pinout 31
8 Firmware Examples 32
8.1 STM32 32
9 Mechanical 33
Ordering Information 35
Content 36
Appendix 37
Revision History 37
Document Revision 37
Chip Revision 37
Documentation Feedback 37
Sales Contact 37
Customization 37
Trademarks 37
Disclaimer 38

Appendix

Revision History

Document Revision

Date	Revision Code	Description
May 2026	ISDOC145**B**	- Correction to DIR and RX/TX pin description and the Pins section. They are open-drain. - Correction to hardware example schematic SCH0068A: the DIR pin was missing a pull-up resistor.
April 2026	ISDOC145**A**	- Initial release.

Chip Revision

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

Date	Revision Code	Description
April 2026	**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.