IS4310: I2C Modbus RTU Slave Stack

500 Holding Registers

Main Advantages

- Eliminates engineering time and costs for protocol implementation and testing.
- Simple and easy to use solution.
- Reduces product time-to-market (TTM).
- Reduces microcontroller CPU load.
- Reduces impact on microcontroller peripherals (no need for timers or UARTs).
- Saves microcontroller pins with a shared I2C.
- Features a small, easy-to-solder SO8N package.
- Provides a low-cost solution.
- Makes the Modbus protocol transparent.
- I2C Speeds: 100kHz, 400kHz, and 1MHz.

Applications

- Modbus Sensors
- Modbus Actuators
- Custom Modbus Devices
- Communications between PCBs

Modbus Stack Characteristics

- 500 Holding Registers
- 4 Configuration Holding Registers:
  - Modbus Address ID
  - Baud Rate
  - Parity Bit
  - Stop Bits
- Implemented Function Codes:
  - 0x03 - Read Holding Registers
  - 0x06 - Write Single Register
  - 0x10 - Write Multiple Registers

General Description

The IS4310 is a chip that integrates a Modbus RTU Slave stack. It features an internal memory of 500 Holding Register. These registers can be accessed for reading and writing both by a microcontroller (via I2C) and by the Modbus Master device (such as PLC, computer, etc.).

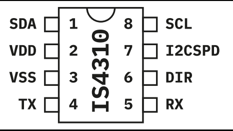
The IS4310 features a UART port that can be connected to the desired electrical interface transceiver: RS485, RS422, RS232, etc.

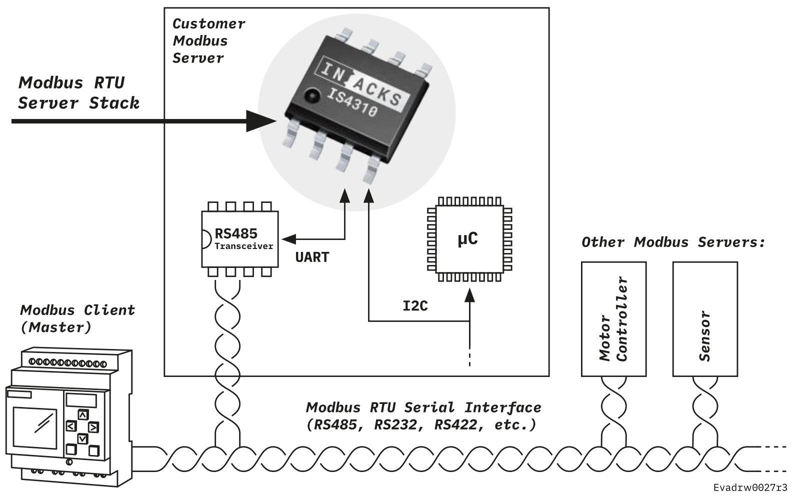
The aim of the IS4310 is to save engineering time and costs associated with implementing and testing the Modbus RTU communication protocol, providing a reliable solution that reduces the time-to-market (TTM) of your product.

The IS4310 also brings benefits to your microcontroller: it utilizes I2C, eliminating the need for dedicated pins since I2C can be shared with other peripherals. Additionally, it eliminates the need for timers and decreases the CPU load on the microcontroller.

The device operates at 3.3V, and its I/O pins are 5V tolerant, allowing the use of either 3.3V or 5V transceivers. It is available in two temperature ranges: Industrial (-40ºC to +85ºC) and Extended (-40ºC to +125ºC).

Part Number	Package	Op. Temperature
IS4310-S8-I	SO8N	-40ºC to +85ºC
IS4310-S8-E	SO8N	-40ºC to +125ºC





Product Selection Guide

Group	Part Number		Form Factor	Physical Layer	Stack	Description
Only Stack	IS4310-S8		SO8N	UART	Modbus RTU Server	Modbus RTU Slave Stack Chip. https://inacks.com/is4310 Contact us for smaller IC packages or product customization.
Stack with Physical Layer	IS4310-485M2		Castellated Holes Module	RS485	Modbus RTU Server	IS4310 with RS485 Transceiver. Industrial communications. https://inacks.com/is4310-485m2
Stack with Physical Layer	IS4310-ISO485M6		Castellated Holes Module	Isolated RS485	Modbus RTU Server	IS4310 with Isolated RS485 Transceiver. The isolation offers more robust communications and longer RS485 bus distances. https://inacks.com/is4310-iso485m6
Stack with Physical Layer	IS4310-232M4		Castellated Holes Module	RS232	Modbus RTU Server	IS4310 with RS232 Transceiver. https://inacks.com/is4310-232m4
Evaluation Boards	Kappa4310Ard		Arduino Compatible	RS485	Modbus RTU Server	IS4310 Evaluation Board with RS485 Transceiver. Compatible with Arduino. https://inacks.com/kappa4310ard
Evaluation Boards	Kappa4310Rasp		Raspberry Pi Compatible	RS485	Modbus RTU Server	IS4310 Evaluation Board with RS485 Transceiver. Compatible with Raspberry Pi. https://inacks.com/kappa4310rasp

1. Electrical Specifications

Absolute Maximum Ratings

Parameter			Min	Max	Unit
Input Voltage	VDD Pin		-0.3	4	V
	SCL Pin		-0.3	5.5	
	SDA Pin		-0.3	5.5	
	RX, TX and DIR Pin		-0.3	5.5	
	I2CSPD Pin		-0.3	4	
Current Sourced/Sunk by any I/O or Control Pin				±20	mA
Temperature	Operating Temperature	IS4310-S8-I	-40	+85	ºC
		IS4310-S8-I	-40	+125	
	Storage Temperature		-65	+150	
Electrostatic Discharge (T_A = 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, RX, TX and DIR Pins	$V_{I/O-IN}$	-0.3	3.3	5.5	
Source/Sink Current at SCL, SDA, RX, TX and DIR 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}	

Flash Memory Cell Characteristics

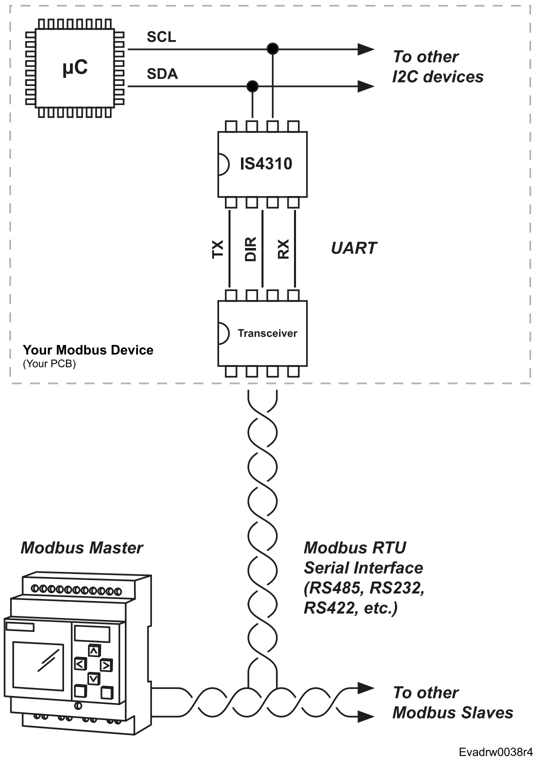
This memory stores the configuration of the Modbus communications: Slave ID (MBADR), Baud Rate (MBBDR), Parity (MBPAR), and Stop Bits (MBSTP).

Parameter		Min	Nom	Max	Unit
Write Cycles		10k	-	-	Cycles
Data Retention	T_A = 85ºC	30	-	-	Years
	T_A = 105ºC	15	-	-	
	T_A = 125ºC	7	-	-	
Programming time		-	-	25	ms

2. Detailed Description

2.1. IS4310 Description

The IS4310 is an integrated circuit running a Modbus RTU Slave Stack, providing an all-in-one and independent Modbus Slave solution ready for integration with a Microcontroller in customer applications.



Note: This schematic is simplified for clarity and should not be used for technical reference.

The IS4310 features two communication buses: a TTL UART for Modbus and an I2C-Serial Interface for the Microcontroller.

The Modbus-UART can connect with different types of transceivers, such as RS485, RS422, RS232, fiber and radio, with RS485 being the most typical. The I2C-Serial Interface can connect to a Microcontroller, microprocessor, single board computer like Raspberry Pi, or development boards like Arduino.

The IC contains 504 Holding Registers (from address 0 to 503), each 16 bits long. Of these, 500 registers are available for the customer's application and utilize RAM as their memory type. Four registers are dedicated to the Modbus configuration: Modbus Address ID, Baud Rate, Parity and Stop Bits.

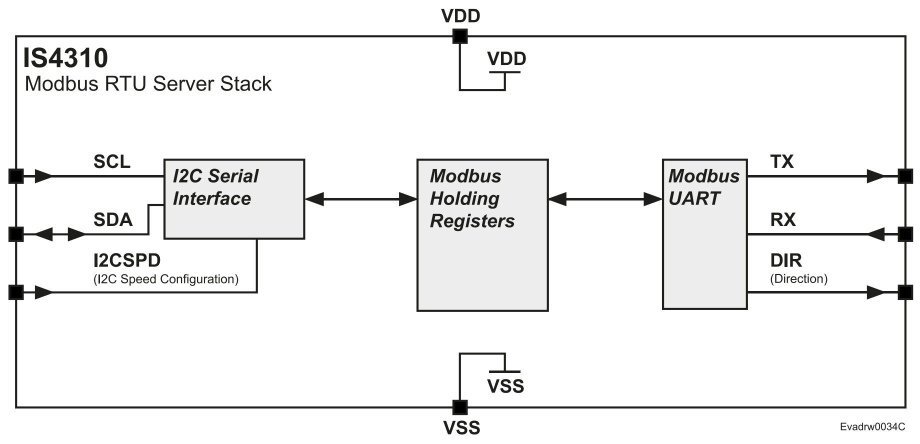
The Holding Registers is where the PLC (Modbus Master) and the Microcontroller (customer application) will exchange information.

In a motor controller application, for example, the PLC can write the desired motor speed to specific Holding Registers and read the motor's power consumption from others. These registers act as a shared memory bank where information is exchanged between the two components of the Modbus communication system: the Master and the Slave.

Essentially, the IS4310 operates as an I2C memory device integrated with the Modbus protocol, featuring a memory map that represents the Holding Registers. This map allows both the PLC and the Microcontroller to read from and write to the registers as needed.

It is available in Industrial (-40ºC to +85ºC) and Extended Temperature Range (-40º to +125ºC).

This company and the products provided herein 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.



2.1.1. Function Codes

Following Function Codes are available on the IS4310 (Modbus Slave):

- 0x03 - Read Holding Registers
- 0x06 - Write Sigle Register
- 0x10 - Write Multiple Registers

If the PLC (Modbus Master) attempts to execute a query with a Function Code not listed above, the IS4310 (Modbus Slave) will respond with Exception Code 1.

2.1.2. Exception Codes

There are scenarios where the PLC can issue invalid queries to the IS4310 (Modbus Slave). In such cases, the IS4310 will respond to the PLC (Modbus Master) with one of the following Modbus Exception Codes:

Code 1: ILLEGAL FUNCTION
The Modbus Master queried a non-valid Function Code to the IS4310 (Modbus Slave). Valid Function codes are 0x03, 0x06, 0x10.

Code 2: ILLEGAL DATA ADDRESS
The Modbus Master attempted to read or write one or more Holding Registers higher than 503.

Alternatively, the query's starting register was valid, but the total number of registers requested exceeded 503.

Code 3: ILLEGAL DATA VALUE
The query performed by the PLC has errors.

2.2. IS4310 Advantages

The use of the IS3410 brings the following benefits:

1. Eliminates engineering time and costs for protocol implementation and testing.
2. Reduces product time-to-market (TTM).
3. Increased product reliability.
4. Saved microcontroller pins.
5. Reduced microcontroller CPU load.

The IS4310 significantly reduces engineering time by eliminating the need to manually implement and test the Modbus protocol. This time saving allows engineers to allocate resources more efficiently towards other critical aspects of product development. Additionally, this efficiency facilitates a faster time-to-market (TTM) and shortens the time to develop a minimum viable product (MVP). The streamlined development process enables companies to accelerate their product launch timelines, meeting market demands swiftly and effectively.

Using the IS4310 solution enhances customer application reliability. With the Modbus protocol already embedded and tested, there are fewer bugs and issues in the customer application, leading to a more robust and reliable product in the field. This also reduces the need for releasing firmware updates to patch undetected bugs and issues related to the Modbus communication protocol.

Additionally, using the IS4310 can reduce Microcontroller pin requirements by saving three dedicated UART pins (Rx, Tx, and direction) and utilizing a shared bus like I2C.

Furthermore, offloading the Modbus protocol processing to the IS3410 saves Microcontroller CPU load, Flash, RAM memory, and TMR resources. This efficiency enhancement allows the Microcontroller to handle other tasks more effectively, contributing to overall system performance improvements and enabling the selection of a lower-end Microcontroller.

In conclusion, the usage of IS4310 not only streamlines development, enhances reliability, and accelerates time-to-market but also optimizes Microcontroller resources, making it a comprehensive solution for efficient product development and deployment.

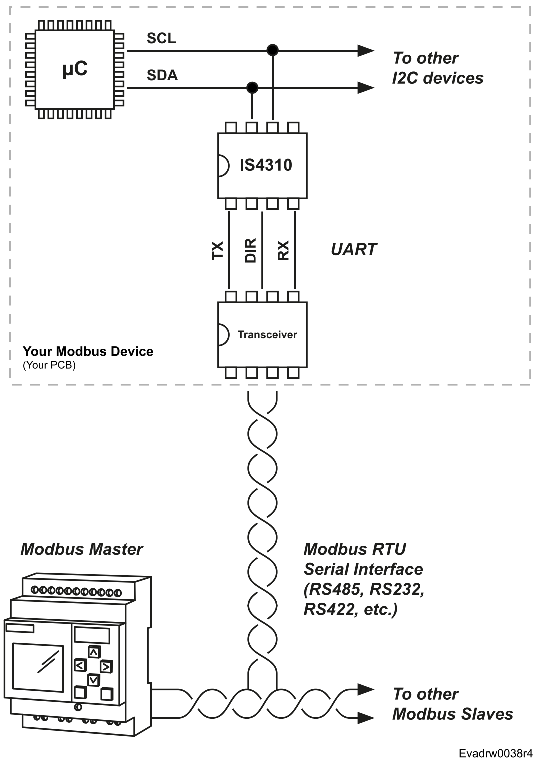
2.3. Modbus UART Port

The IS4310 is compatible with any Modbus RTU Serial Interface, including RS485, RS422, RS232, and others, thanks to its UART port. A transceiver matching the serial interface of the field bus (RS485, RS422, RS232, etc.) must be connected to the IS4310 UART port. This transceiver adapts the field bus voltage levels to 3.3V or 5V, ensuring proper operation with the IS4310.

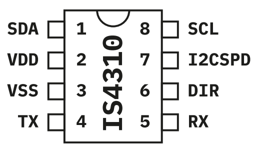
Note: Connecting field buses like RS485 or others directly to the IS4310 will not work and will permanently damage the device.

For example, if the customer application connects to an RS485 field bus, an RS485 transceiver such as the THVD1330 should be used.

Refer to chapter “Implementation Guide” for hardware design example.



3. Pin Description



Pin	Name	Type	Description
1	SDA	Open Drain 5V Tolerant	I2C-compatible 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	Modbus UART pins in TTL voltage levels. TX is the IS4310 transmit pin, RX the IS4310 receive pin. Attention: Only digital 3.3V or 5V can be applied to this pin. Use the appropriate transceiver to connect the IS4310 with the desired bus. Do not connect field buses such as RS485, RS422, RS232 or other directly to this pin. Field buses voltages are not compatible with TTL.
5	RX	Digital Input 5V Tolerant	
6	DIR	Digital Output Push-Pull	Direction pin for the transceivers, used to control the data flow direction on the bus. This pin goes high only when the IS4310 is transmitting data. It goes low while receiving data or waiting for data. Example: In an RS485 transceiver, the Receiver Output Enable (RE) and Driver Output Enable (DE) pins are connected to this 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). Attention: Voltage above 4V will damage the device.
8	SCL	Open Drain 5V Tolerant	I2C-compatible Clock pin. Open drain, it requires pull-up.

TX and RX Pins

Modbus UART Transmit and Receive Pins.

These pins handle UART transmit and receive functions for Modbus data and operate at TTL levels of 3.3V and they are 5V tolerant.

To interface with the field bus, these pins must connect to a suitable transceiver based on the field bus used: RS485, RS422, RS232, or others.

Please note that applying directly field bus (RS485, RS422, RS232, etc.) voltage levels to those pins will permanently damage the device.

For an RS485 fieldbus, use an RS485 transceiver, such as the THVD1330DR, to convert RS485 differential signaling to TTL/CMOS voltage levels. For an RS232 fieldbus, a transceiver like the MAX3221 can be used. Refer to the “Hardware Implementation ” chapter for more details.

DIR Pin

Modbus Direction Pin.

This pin is typically used in transceivers to control the data flow (sending or receiving). For RS485 transceivers, it connects to the DE and $\overline{\text{RE}}$ pins of the transceiver.

Modbus Over Serial Line is usually implemented on “Two-Wire” RS485 electrical interface, which operates in a half-duplex topology. Therefore, a direction pin is needed to indicate whether the transceiver should send or receive data. By default, the DIR pin is in a low state, which sets the transceiver to receive mode.

SCL and SDA Pins

I2C-Compatible Bus Interface Pins.

SCL (Serial Clock Line): This pin is used to synchronize data transfer between the IS4310 device and the microcontroller or other CPU.

SDA (Serial Data Line): This bidirectional pin is used for both sending and receiving data between the IS4310 and the microcontroller or other CPU.

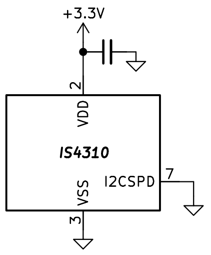
Both pins are open-drain and must be pulled up to 3.3V or 5V. The pull-up resistor value should be chosen based on the bus speed and capacitance. Typical values are 4.7kΩ for Standard Mode (100kbps) and 2kΩ for Fast Mode (400kbps) at both 3.3V and 5V.

I2CSPD Pin

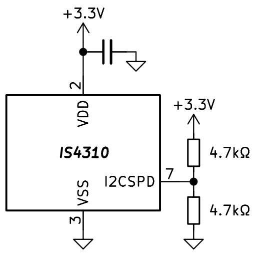
I2C-Serial Interface Speed Selection Pin.

This pin configures the IS4310 internal I2C-Serial Interface timings and filters to properly work with the selected bus speed.

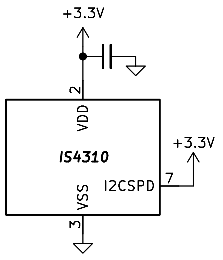
For a **100 kHz** setting, set the I2CSPD pin to VSS.



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.

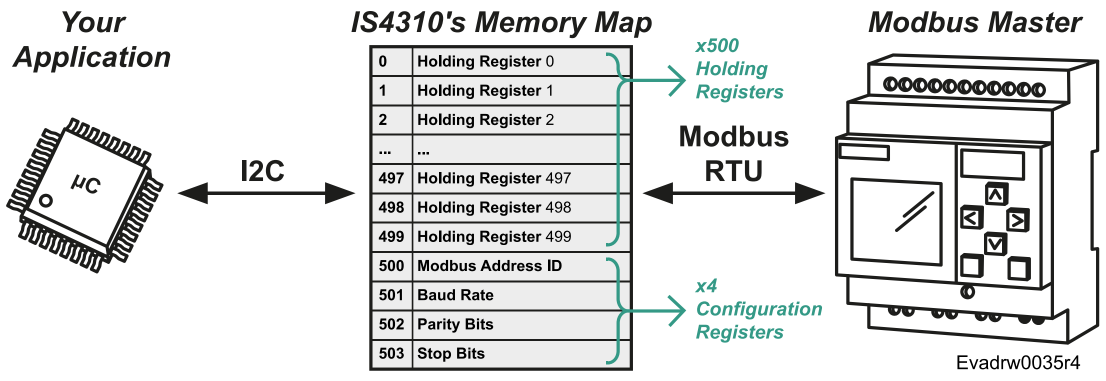
A mismatch between the configured I²C speed and the actual operating I²C speed (e.g., setting I2CSPD to GND for 100 kHz but operating at 1 MHz) can lead to an inconsistent state where some I2C messages are processed while others are not.

Ensure a proper match between the actual operating speed and the configured speed at the I2CSPD pin: If your bus works at 100kHz, ensure the I2CSPD pin is tied to VSS. If it works at 400kHz ensure the pin is at 1.65V. If it works at 1000MHz, ensure the pin is at 3.3V.

4. Memory Description

4.1. Memory Map Organization

The IS4310 is organized internally as a single page containing 504 registers, with addresses ranging from 0 to 503. These registers can be accessed individually or in blocks. Each register is 16 bits long, and there are two types: Holding Registers and Configuration Registers. Both types are readable and writable and can be accessed by the Microcontroller via I2C or by the Modbus Master through a field bus (RS485, RS422, RS232, etc.).



Holding Registers
The Holding Registers consist of 500 volatile RAM registers, with addresses ranging from 0 to 499.

The Holding Registers (HOLDx) are available for your application. For example, if you are developing a gas sensor, these registers can store data such as gas concentration and the total operating hours of the sensor. The Microcontroller will continuously write to these registers, while the Modbus Master will read from them.

If you are developing an actuator, such as a relay module, these registers can store the state of the relays. In this case, the Microcontroller will continuously read from the registers, while the Modbus Master will write to them as needed.

You can also combine reading and writing operations within the same application. For example, when developing a Modbus motor controller, the Modbus Master can write the motor's speed and read its power consumption. In this case, the Microcontroller will continuously read from the registers assigned to the speed and write to the registers related to power consumption.

Configuration Registers
The Configuration Registers consist of four registers, with addresses ranging from 500 to 503. Any modifications to these registers are stored in the internal non-volatile memory. Upon power-up, the IS4310 retrieves the last saved configuration.

Four configuration registers are used to set the Modbus communication parameters: Modbus Address ID (MBADR), Baud Rate (MBBDR), Parity Bit (MBPAR), and Stop Bit (MBSTP).

Both the Modbus Master and the Microcontroller can write to these registers, and the changes take effect immediately.

Important Remark:
In a Modbus network, two slaves cannot have the same Address ID. Doing so will cause both devices to become unresponsive.

4.2. Memory Map Table

I2C Register Address	Modbus Register Address	Register Name	Register Description
0	0	HOLD0	Holding Register 0
1	1	HOLD1	Holding Register 1
2	2	HOLD2	Holding Register 2
3	3	HOLD3	Holding Register 3
4	4	HOLD4	Holding Register 4
5	5	HOLD5	Holding Register 5
…	…	(HOLD6 to HOLD493)	…
494	494	HOLD494	Holding Register 494
495	495	HOLD495	Holding Register 495
496	496	HOLD496	Holding Register 496
497	497	HOLD497	Holding Register 497
498	498	HOLD498	Holding Register 498
499	499	HOLD499	Holding Register 499
500	500	MBADD	Configuration Register, Slave Address Configuration
501	501	MBBDR	Configuration Register, Baud Rate Configuration
502	502	MBPAR	Configuration Register, Parity Bit Configuration
503	503	MBSTP	Configuration Register, Stop Bits Configuration

4.3. HOLDx Registers

HOLDx Registers are Modbus Holding Registers available for use in your application. They can be accessed (read and written) by both the Modbus Master and the Microcontroller. The access can be individually or in block. Each register is 16 bits long and they are volatile RAM.

If your product is a sensor, these registers are typically written by the Microcontroller with sensed data, such as pressure, temperature, or humidity, and read by the Modbus Master.

If the product is an actuator, these registers are usually written by the Modbus Master with control data, such as relay states (on/off), motor speed, or solenoid valve positions, and read by the Microcontroller.

These registers can also serve as a bidirectional data exchange point, allowing both the Modbus Master and the Microcontroller to read and write data.

.

Name:	HOLDx
Description:	Modbus Holding Registers
Address Range:	0 to 499 (0x000 to 0x1F3)
Default value:	0 (0x0000)
Memory Type:	Volatile RAM
Allowed values:	0 to 65535 (0x00 to 0xFFFF)

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
HOLDx [15 to 8]							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
HOLDx [7 to 0]							

4.4. MBADR Register

This register represents the Slave Address of the device on the Modbus network. The value written to this register serves as the identifier for the Modbus device.

The default value is 1. Each Slave must have a unique address, as two Slaves with the same address will cause both devices to become unresponsive.

The allowed address range is from 1 to 247. Attempting to write an address outside this range will have no effect.

This register is placed at the end of the Holding Register section to minimize the risk of accidentally overwriting the Modbus configuration registers.

Any modifications to this register will be saved in the internal non-volatile memory, which supports up to 10,000 cycles. A write cycle occurs only if the written data differs from the previous value. The write process takes 25ms. Upon power-up, the IS4310 automatically retrieves the last saved configuration.

Name:	MBADR
Description:	Slave Address Configuration
Register Address:	500 (0x1F4)
Default value:	1 (0x0001)
Memory Type:	Non-Volatile RAM
Allowed values:	1 to 247 (0x0000 to 0x00F7)

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
-	-	-	-	-	-	-	-

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
MBADR [7 to 0]							

4.5. MBBDR Register

The MBBDR register stores the Modbus Baud Rate configuration. The default value is 113, representing 19200 bps, which is the default Modbus speed.

Allowed configuration values range from 110 to 115; attempting to write any other values will not have any effect.

- Value 110 sets a Modbus speed of 1200bps.
- Value 111 sets a Modbus speed of 2400bps.
- Value 112 sets a Modbus speed of 9600bps.
- Value 113 (default) sets a Modbus speed of 19200bps.
- Value 114 sets a Modbus speed of 57600bps.
- Value 115 sets a Modbus speed of 115200bps.

This register is placed at the end of the Holding Register section to minimize the risk of accidentally overwriting the Modbus configuration registers.

Any modifications to this register will be saved in the internal non-volatile memory, which supports up to 10,000 cycles. A write cycle occurs only if the written data differs from the previous value. The write process takes 25ms. Upon power-up, the IS4310 automatically retrieves the last saved configuration.

Name:	MBBDR
Description:	Baud Rate Configuration
Register Address:	501 (0x1F5)
Default value:	113 (0x0071)
Memory Type:	Non-Volatile RAM
Allowed values:	110 to 115 (0x006E to 0x0073)

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
-	-	-	-	-	-	-	-

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
MBBDR [7 to 0]							

4.6. MBPAR Register

The MBPAR register stores the Modbus Parity Bit configuration. The default value is 122, representing Even Parity, which is de default Modbus Parity.

Allowed configuration values range from 110 to 115; attempting to write any other values will not have any effect.

- Value 120 represents No Parity.
- Value 121 represents Odd Parity.
- Value 122 (default) represents Even Parity.

This register is placed at the end of the Holding Register section to minimize the risk of accidentally overwriting the Modbus configuration registers.

Any modifications to this register will be saved in the internal non-volatile memory, which supports up to 10,000 cycles. A write cycle occurs only if the written data differs from the previous value. The write process takes 25ms. Upon power-up, the IS4310 automatically retrieves the last saved configuration.

Name:	MBPAR
Description:	Parity Bit Configuration
Register Address:	502 (0x1F6)
Default value:	122 (0x007A)
Memory Type:	Non-Volatile RAM
Allowed values:	120 to 122 (0x0078 to 0x007A)

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
-	-	-	-	-	-	-	-

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
-	MBPAR [6 to 0]						

4.7. MBSTP Register

The MBSTP register contains the Modbus Parity Bit configuration. The default value is 131, representing One Stop Bit, which is the default Modbus Sop Bit.

Allowed configuration values range from 131 to 132; attempting to write any other values will not have any effect.

- Value 131 (default) One Stop bit (default).
- Value 132 Two Stop bit.

This register is placed at the end of the Holding Register section to minimize the risk of accidentally overwriting the Modbus configuration registers.

Any modifications to this register will be saved in the internal non-volatile memory, which supports up to 10,000 cycles. A write cycle occurs only if the written data differs from the previous value. The write process takes 25ms. Upon power-up, the IS4310 automatically retrieves the last saved configuration.

Name:	MBSTP
Description:	Stop Bits Configuration
Register Address:	503 (0x1F7)
Default value:	131 (0x0083)
Memory Type:	Non-Volatile RAM
Allowed values:	131 and 132 (0x0083 and 0x0084)

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
-	-	-	-	-	-	-	-

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
MBSTP [7 to 0]							

5. I2C Description

The IS4310 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 and write operations to the IS4310 (I2C-Slave device).

The IS4310 is represented on the bus by the I2C device address: `17` (`0x11`).

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 IS4310's 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 addressing (pointer register) is 16 bits wide, which means 2 bytes are required to set the target address for any I2C read or write operation. The memory map itself contains 504 registers, each 16 bits wide. Therefore, 2 bytes of data are needed to read from or write to each IS4310 register.

The operability of the Read and Write commands of the IS4310 is very similar to an EEPROM memory. Thinking of the IS4310 as an EEPROM memory is a good analogy to quickly understand how to communicate with its memory map.

5.1. Highlights

- **I2C Device Address**: `17 (0x11)`

- **I2C Memory Map Register and Addressing Size**: 16 bits, composed of two bytes — first the MSB, then the LSB.

- **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 Plus (1 MHz), recommended SCL and SDA pull-up value: 2 kΩ

- **I2C Supported Operations**:
  - Single-Byte Write
  - Multiple-Byte Write (up to 3,601 registers)
  - Single-Byte Read
  - Multiple-Byte Read (up to 3,601 registers)

- **Overreading and Overwriting the memory**:
  - If a write operation starts at a valid memory address (`0` to `3599`) 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 `3599`) will result in a NACK and data will be discarded.
  - If a read operation starts at a valid memory address (`0` to `3599`) 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 `3599`) will return a value of `0xFFFF`.

5.2. Read Operations

5.2.1. Single Word Read

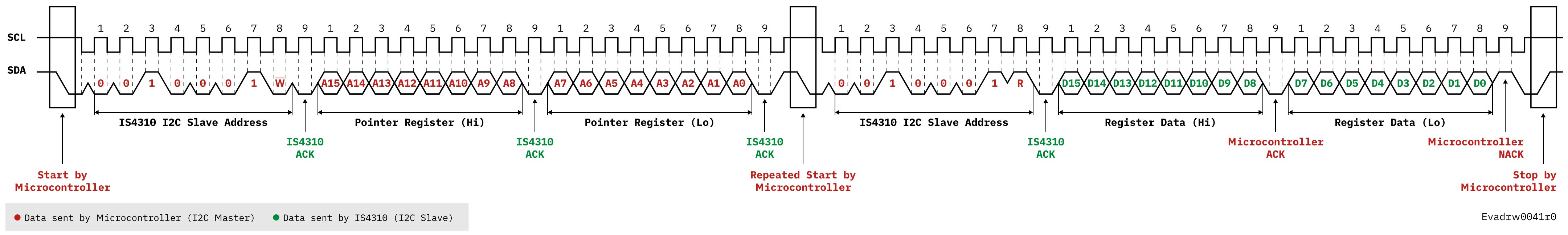
Reading a single word is an action performed by the Microcontroller (I2C-Master) to access any register within the IS4310 memory (I2C-Slave), regardless of the last read or written position. To perform this action, the microcontroller must first load the address of the IS4310 register to be read into the IS4310's internal Pointer Register. Once the address is set, the microcontroller can retrieve the data from the specified register.

To initiate the Single Word Read operation, the microcontroller begins by pulling down the SDA while the SCL is high to create a Start Condition. It then sends the IS4310 I2C device address (0x11) with the R/W bit set to '0' (write). Upon receiving the device address, the IS4310 acknowledges it. Subsequently, the 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 IS4310. This sets the address of the next word to be read in the Pointer Register.

Next, the content of the Pointer Register, which is a word (two bytes), needs to be read.

The microcontroller generates a Repeated Start Condition, followed by the IS4310 I2C device address (0x11) with the R/W bit set to '1' (read), instructing the IS4310 to retrieve data. The IS4310 acknowledges and responds with the most significant byte, which the microcontroller acknowledges. Then, the IS4310 sends the less significant byte, 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 IS4310 goes from addresses 0 to 503. If a Read Operation is performed with a Pointer Register higher than 503, the read result will be 0xFFFF.



5.2.2. Multiple Word Read

Multiple Word Read functions similarly to Single Word Read but can read a block of up to 500 registers in a single operation. Remember, the registers are 16-bit words consisting of 2 bytes, so the number of registers retrieved should always be even.

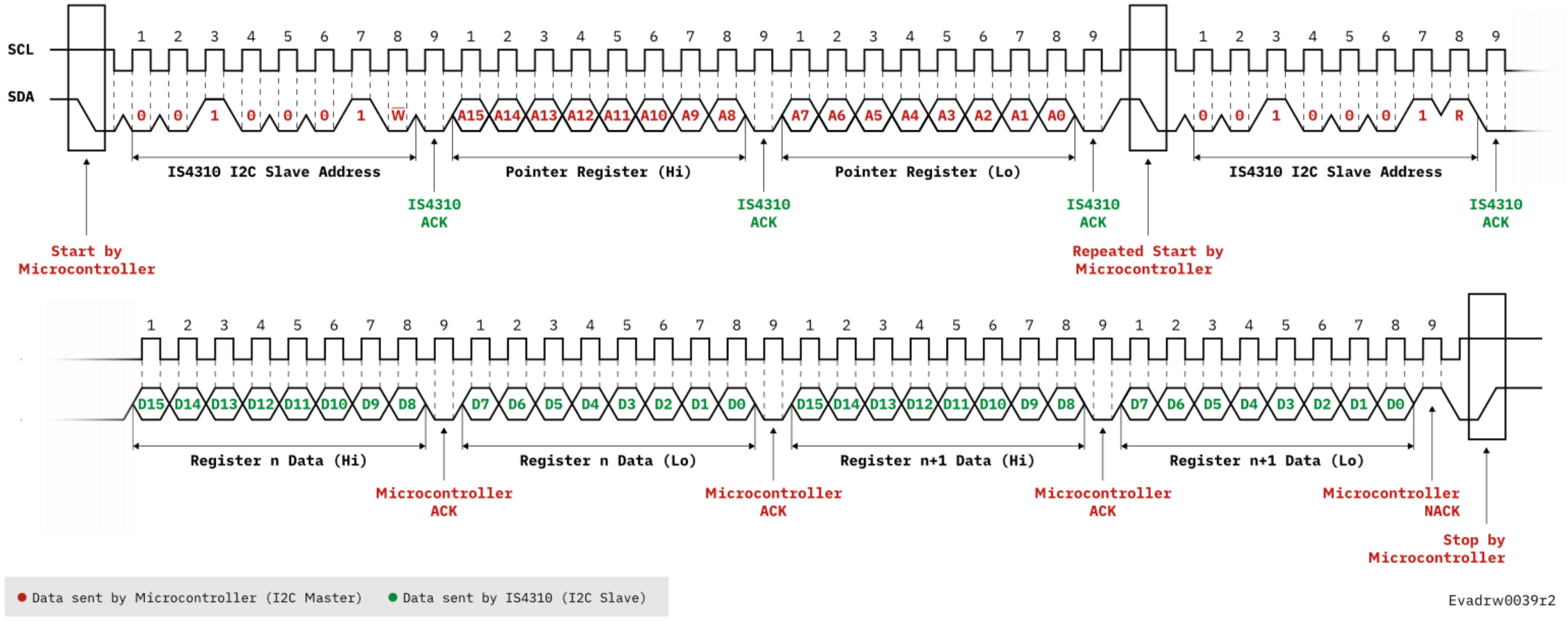
To perform a Multiple Word Read, follow the same procedure as for a Single Word Read until the first data word is received. After receiving the first word, instead of generating a Not Acknowledge (NACK), the microcontroller should continue acknowledging (ACK) each received data byte from the IS4310 for as many words as it intends to read. To conclude the read operation, after reading the last data word, the microcontroller should generate a Not Acknowledge (NACK) and a Stop Condition.

With each word read, the Pointer Register increments by one.

Invalid Memory Addressing
The valid memory range of the IS4310 goes from addresses 0 to 503.

If the Read Operation is performed with a Pointer Register within the valid memory range (0 to 503), but the data retrieval extends beyond register 503, a rollover to position 0 will occur. For example, the value of register 504 will correspond to the content of register 0.

If a Read Operation is performed with a Pointer Register value higher than 503, the read result will be 0xFFFF.



5.3. Write Operations

5.3.1. Single Word Write

Writing a single word is an action performed by the Microcontroller (I2C-Master) to write data to any register within the IS4310 memory (I2C-Slave), regardless of the last read or written position. To perform this action, the Microcontroller must first load the address of the IS4310 register to be written into the IS4310's internal Pointer Register. Once the address is set, the Microcontroller can send the data to be stored.

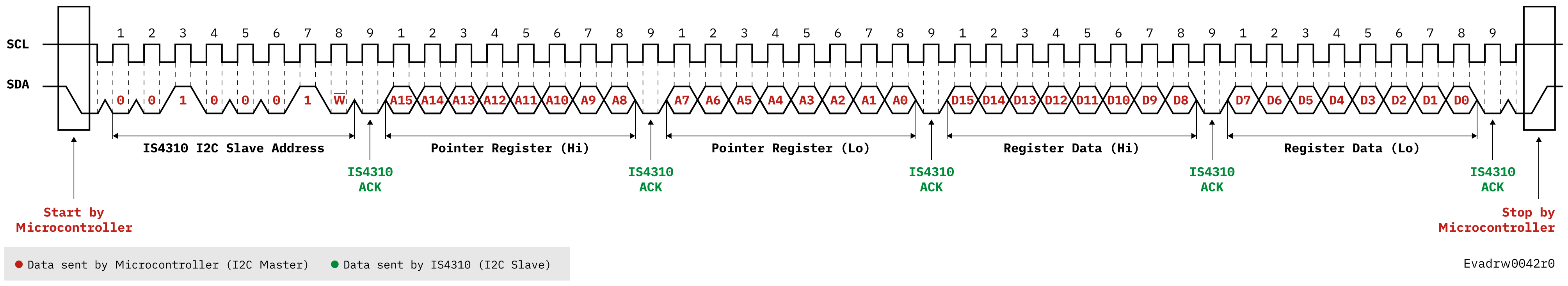
To initiate the Single Word Write operation, the Microcontroller begins by pulling down the SDA line while the SCL line is high, creating a Start Condition. It then sends the IS4310 I2C device address (0x11) with the R/W bit set to '0' (write). Upon receiving the device address, the IS4310 acknowledges it. Subsequently, the Microcontroller sends the two bytes of the Pointer Register address: the most significant byte first, followed by the least significant byte, each acknowledged by the IS4310. This sets the address of the next word to be written in the Pointer Register, preparing the device to receive the data.

The Microcontroller then sends the most significant byte of the word to be written first, which the IS4310 acknowledges. The Microcontroller follows by sending the least significant byte of the word, which the IS4310 also acknowledges. Finally, the Microcontroller issues a Stop Condition by raising the SDA line while the SCL line is high.

After the Stop Condition, if any of the Modbus Configuration Registers (MBADR, MBBDR, MBPAR, MBSTP) are written with a value different from the previous one, a 25 millisecond Flash Memory write cycle will begin.

Invalid Memory Addressing

The valid memory range of the IS4310 goes from addresses 0 to 503. If a Write Operation is performed with a Pointer Register higher than 503, the IS4310 will answer with a NACK on the first received byte of the word.



5.3.2. Multiple Word Write

A Multiple Word Write performs a similar operation to a Single Word Write, but instead of writing to only one register, it can write to a block of up to 500 registers in a single operation.

To perform a Multiple Word Write, follow the same procedure as for a Single Word Write until the first data word is received. After receiving the first word, instead of generating a Stop Condition, the Microcontroller should continue sending data words. To conclude the write operation, after sending the last data word, the Microcontroller should generate a Stop Condition.

With each word written, the Pointer Register increments by one.

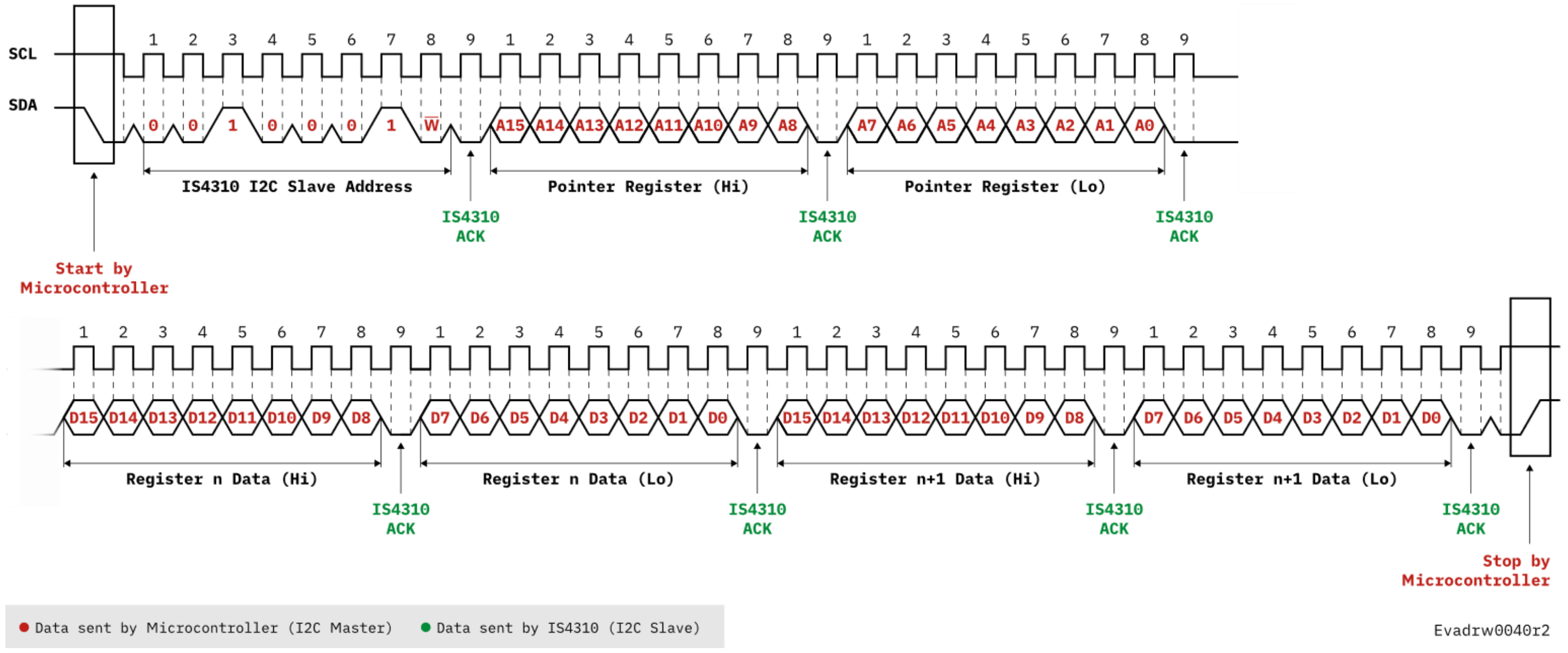
After the Stop Condition, if any of the Modbus Configuration Registers (MBADR, MBBDR, MBPAR, MBSTP) are written with a value different from the previous one, a 25 millisecond Flash Memory write cycle will begin.

Invalid Memory Addressing

The valid memory range of the IS4310 goes from addresses 0 to 503.

If a Write Operation is performed with a Pointer Register within the valid memory range (0 to 503) but exceeds the last memory register (503), a rollover to position 0 will occur. For example, writing a value to register 504 will result in writing the value to register 0.

If a Write Operation is performed with a Pointer Register higher than 503, the IS4310 will answer with a NACK on the first received byte of the word.



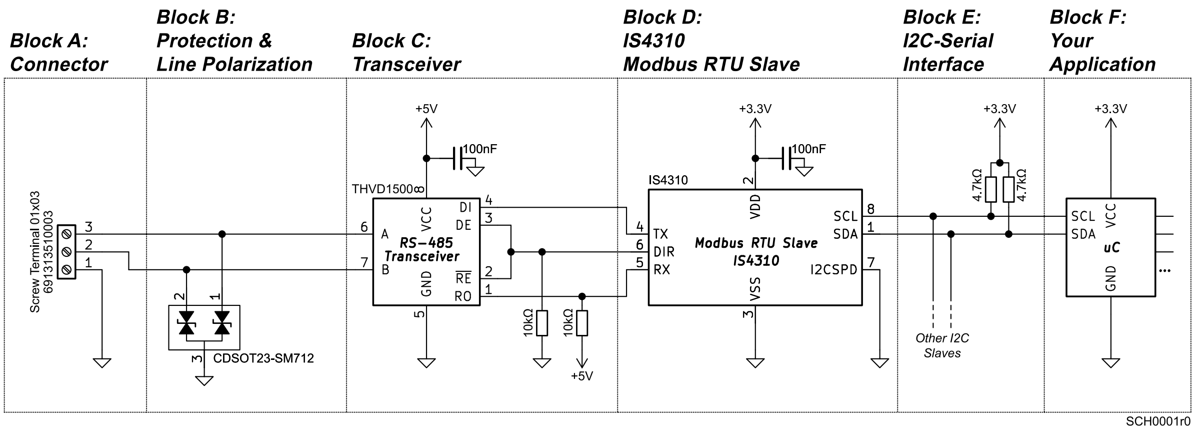
6. Hardware Implementation Guide

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 Modbus specifications.

6.1. RS485 Example

This example shows the design of a Modbus over Serial Line working in RS485.

More examples can be found on the website.



Block A: Connector

Typical Modbus Serial Line connectors include Screw Terminals, RJ45, and D-Sub 9-pin (commonly known as DB9), among others. The device-side connector must be female, while the cable-side connector must be male.

The recommended connector is RJ45, but in the schematic, a screw terminal is used for simplicity. When selecting a connector, always choose the shielded version if available. RJ45 and DB9 connectors typically come with shielded options, while terminal blocks usually do not.

On the cable-side connector, make sure to connect the cable shield to the connector shield to ensure proper electrical continuity across all cable shields on the bus.

Do not connect the shield to the Common. All cable shields should be connected to Common and Protective Ground at a single point for the entire bus, ideally at the master device.

In the example, the connector has three positions: A, B, and Common. A and B are the differential lines for the transceiver, while Common serves as the reference point for the A and B signals. Common must be connected to the GND of your circuit.

Optionally, power can be supplied to your system through the Modbus connector. In this case, a four-position connector would be used for A, B, Common, and Power. In that case, the Common serves as the reference for A and B signals as well as the return path for Power. The voltage should be within the 5V to 24V range.

Block B: Protection & Line Polarization

Protection
The protection stage is influenced by several factors, including the intrinsic robustness and protection features of the transceiver, the potential harshness of the fieldbus environment, 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 are used to protect against surge transients.

Line Polarization
Line Polarization is the process of biasing the RS485 bus to a known state by pulling signal A down and pulling signal B to 5V using resistors in the range of 450 to 650Ω. This ensures that the bus has a defined idle state.

When there is no data activity on an RS-485 balanced pair, the lines are not actively driven and are therefore susceptible to external noise or interference. To ensure that the transceiver remains in a stable state when no data signal is present, some transceivers require a biasing circuit. However, not all transceivers need this.

When selecting your transceiver, confirm in the datasheet whether line polarization is necessary or not. If it is necessary, you must document it in the product guide.

If polarization is needed, it should ONLY be implemented at one location on the bus, typically at the master device.

Bus polarization is a good technic to increase the resistance of the bus to external noise or interferences. However, it has the drawback of significantly reducing the number of devices that can support the bus.

Block C: Transceiver

Modbus over Serial Line typically employs the RS485 electrical interface, which uses a transceiver to adapt RS485 fieldbus voltage levels to TTL voltage levels for the IS4310. Other electrical interfaces such as RS422 or RS232 can also be utilized.

A pull-down resistor on DE and RE will keep the transceiver in ‘receiver’ state by default, ensuring it does not disturb the fieldbus. Pull-up resistor on RO will keep the RX line clear.

Using a 5V transceiver is a good technic to increase the resistance of the bus to external noise or interferences. 5V transceivers can be used with the IS4310 since TX, RX and DIR pins are 5V tolerant.

Block D: IS4310 Modbus RTU Slave

The IS4310 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 100nF, 10-25V low-ESR ceramic capacitor.

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

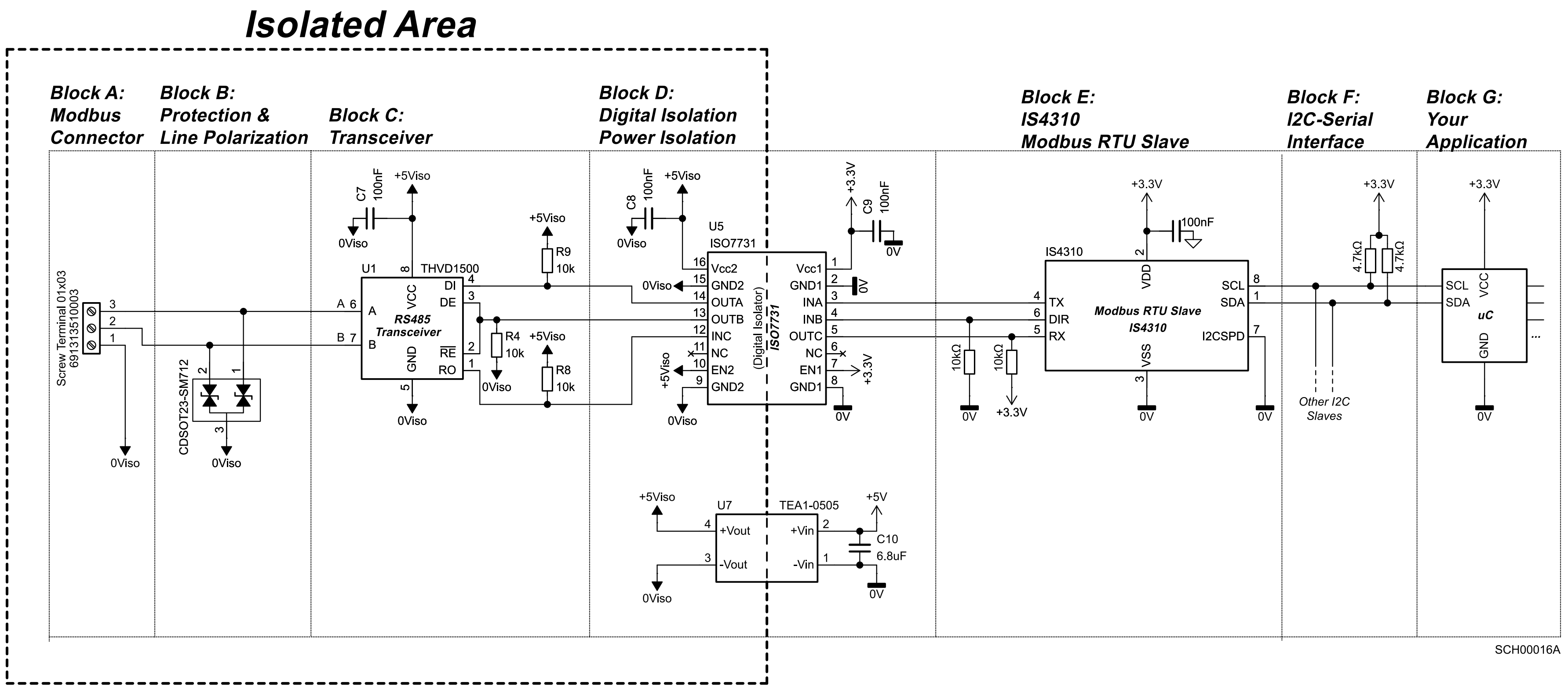
Block E: I2C-Serial Interface

For proper operation of the I2C Serial Interface, pull-up resistors to 3.3V or 5V are necessary. Typical resistor values are 4.7kΩ for Standard Mode (100kHz) and 2.2kΩ for both Fast Mode (400kHz) and Fast Mode Plus (1MHz).

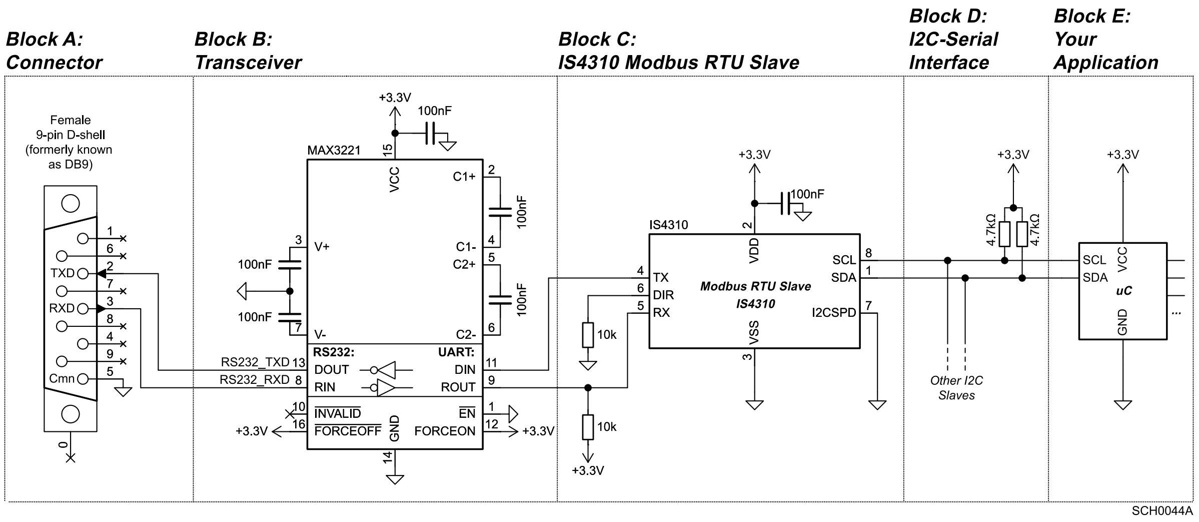
Block F: Your Application

Here is the rest of your product design. Typically, a microcontroller interfaces with the IS4310, 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.

6.2. Isolated RS485 Example

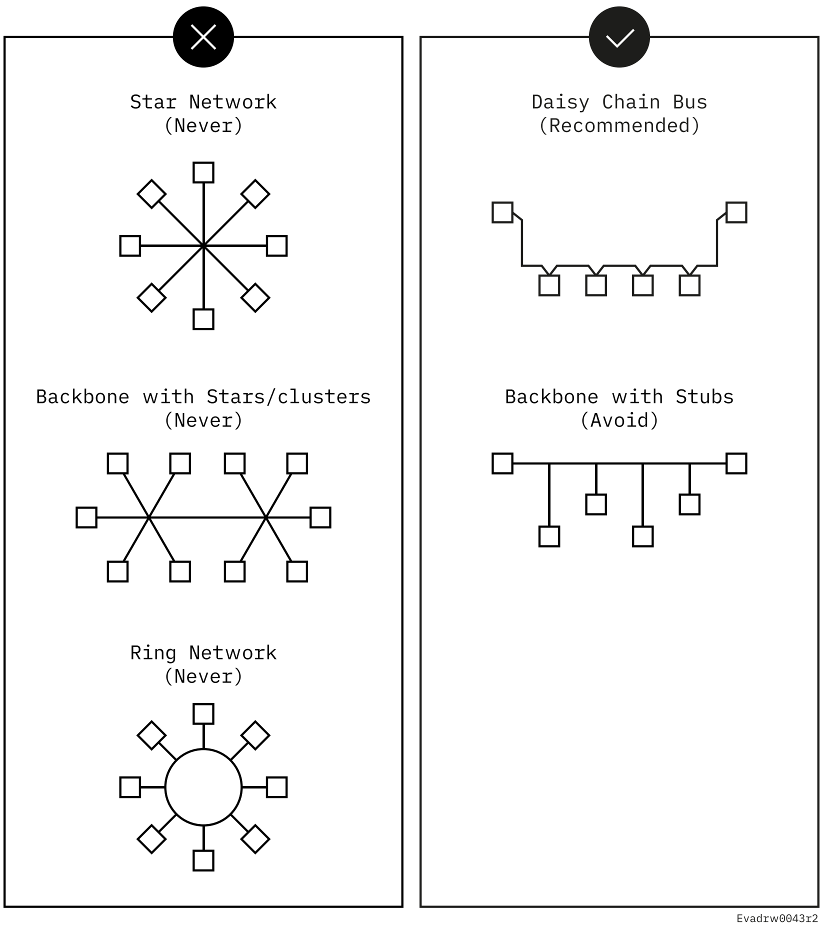

Isolated Area
Block A: Modbus Connector
Block B: Protection & Line Polarization
Block C: Transceiver
Block D: Digital Isolation Power Isolation
Block E: IS4310 Modbus RTU Slave
Block F: I2C-Serial Interface
Block G: Your Application
Screw Terminal 01x03 691313510003
CDSOT23-SM712
U1 THVD1500
RS485 Transceiver
R9 10k
R4 10k
R8 10k
C7 100nF
C8 100nF
C9 100nF
U5 ISO7731
(Digital Isolator) ISO7731
U7 TEA1-0505
C10 6.8uF
IS4310
Modbus RTU Slave IS4310
4.7kΩ
10kΩ
uC
Other I2C Slaves
+5Viso
0Viso
+3.3V
0V
+5V
SCH00016A

6.3. RS232 Example

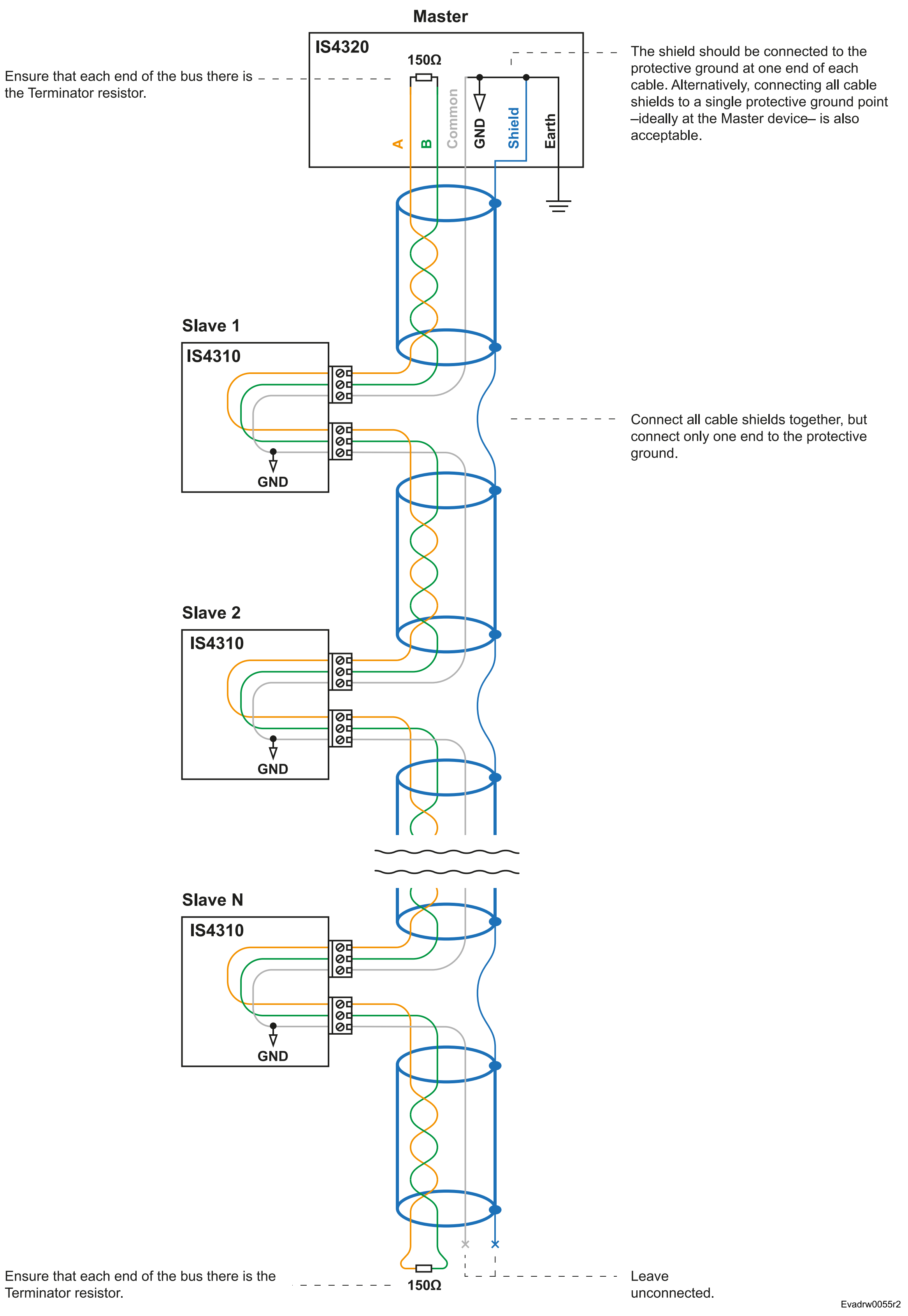


6.4. Bus Topology

In an RS485 setup without a repeater, a single trunk cable runs through the system, with devices connected in a daisy-chain manner. Short cables derivations (stubs) are also allowed but not recommended. Keep the derivation distance as short as possible. Other topologies are not allowed.



6.5. Cable Wiring


Master
IS4320
150Ω
A
B
Common
GND
Shield
Earth
Ensure that each end of the bus there is the Terminator resistor.
The shield should be connected to the protective ground at one end of each cable. Alternatively, connecting all cable shields to a single protective ground point –ideally at the Master device– is also acceptable.
Slave 1
IS4310
GND
Connect all cable shields together, but connect only one end to the protective ground.
Slave 2
IS4310
GND
Slave N
IS4310
GND
Ensure that each end of the bus there is the Terminator resistor.
150Ω
Leave unconnected.
Evadrw0055r2

7. Firmware Implementation Guide

The following chapter presents firmware examples for different platforms for demonstration purposes only and is not part of the product standard. Customers must develop their own firmware, perform all necessary tests, and validate the final product according to applicable regulations and Modbus specifications.

7.1. Arduino Example

Coding for the IS4310 requires no dedicated library, making it easy to maintain and port to new Arduino boards or other microcontrollers.

This code reads the Modbus Slave ID and prints it to the terminal. Then, it stores a humidity variable in Modbus Holding Register address 0. This variable can be accessed by a Modbus Master device, such as a PC, PLC, or other controller.

You can download the Arduino project from the IS4310 product page.

This example uses the Kappa4310Ard Evaluation Board. Check the Kappa4310Ard product folder for more information.

```
#include <Wire.h>

void writeHoldingRegister(uint16_t holdingRegisterAddress, uint16_t data) {
  Wire.beginTransmission(0x11); // This is the I2C Chip Address of the IS4310. Never changes.
  // A Holding Register address is 16-bits long, so we need to write 2 bytes to indicate the address.
  Wire.write((holdingRegisterAddress >> 8) & 0xFF); // Send high 8-bits of the Holding Register Address we want to write.
  Wire.write(holdingRegisterAddress & 0xFF); // Send low 8-bits of the Holding Register Address we want to write.
  // A Holding Register data register is 16-bits long. So we need to write 2 bytes to make a full Holding Register Write:
  Wire.write((data >> 8) & 0xFF); // Send high 8-bits of the data we want to write to the Holding Register.
  Wire.write(data & 0xFF); // Send low 8-bits of the data we want to write to the Holding Register.
  Wire.endTransmission();
}

uint16_t readHoldingRegister(uint16_t holdingRegisterAddress) {
  uint16_t result; // This is the variable where the read data will be saved.
  Wire.beginTransmission(0x11); // This is the I2C Chip Address of the IS4310. Never changes.
  // A Holding Register address is 16-bits long, so we need to write 2 bytes to indicate the address.
  Wire.write((holdingRegisterAddress >> 8) & 0xFF);  // Send high 8-bits of the Holding Register Address we want to read.
  Wire.write(holdingRegisterAddress & 0xFF);         // Send low 8-bits of the Holding Register Address we want to read.
  Wire.endTransmission(false);
  // A Holding Register data register is 16-bits long. So we need to read 2 bytes to make a full Holding Register Read:
  Wire.requestFrom(0x11, 2);  // From the IS4310, request 2 bytes (2 bytes make a full Holding Register).
  result = Wire.read(); // Read the first byte.
  result = result << 8; // Make space for the second byte.
  result = result | Wire.read(); // Read the second byte.
  return result; // Return the read 16-bit register.
}

void setup() {
  uint16_t ModbusSlaveID;

  Wire.begin(); // Initialize the I2C.
  Serial.begin(9600); // Initialize  the Serial for the prints.

  // The Modbus Slave ID is stored in the Holding Register Address 500 of the IS4310, let's read it:
  ModbusSlaveID = readHoldingRegister(500);

  // Let's print the read Modbus Slave ID:
  Serial.println("");
  Serial.print("The Modbus Slave Address is: ");
  Serial.println(ModbusSlaveID);
}

void loop() {
  uint16_t humidity = 47; // Let's imagine a humidity sensor that reads a level of 47% RH.
  // Let's write the humidity to the Holding Register Address 0:
  writeHoldingRegister(0, humidity);
  delay(1000);
}
```

7.2. STM32 Example

Coding for the IS4310 requires no dedicated library, making it easy to maintain and port to new STM32 or other microcontrollers

The following code is an abstraction of the main.c file from the ISXMPL4310ex9 example. All external HAL routines and function calls have been removed for explanation proposals.

This example demonstrates:

1. How to read a potentiometer (simulating a sensor) and store its state in Holding Register 0.
2. How to control an RGB LED (simulating an actuator) using GPIO pins based on values in Holding Registers 1, 2, and 3.

You can download the full STM32 project from the IS4310 product page.

This example uses the Kappa4310Ard Evaluation Board. Check the Kappa4310Ard product folder for more information.

```
uint16_t readHoldingRegister(uint16_t registerAdressToRead) {
    uint8_t IS4310_I2C_Chip_Address; // This variable stores the I2C chip address of the IS4310.
    IS4310_I2C_Chip_Address = 0x11; // The IS4310's I2C address is 0x11.
    // The STM32 HAL I2C library requires the I2C address to be shifted left by one bit.
    // Let's shift the IS4310 I2C address accordingly:
    IS4310_I2C_Chip_Address = IS4310_I2C_Chip_Address << 1;

    // The following array will store the read data.
    // Since each holding register is 16 bits long, reading one register requires reading 2 bytes.
    uint8_t readResultArray[2];

    // This variable will contain the final result:
    uint16_t readResult;

    /*
     * This is the HAL function to read from an I2C memory device. The IS4310 is designed to operate as an I2C memory.
     *
     * HAL_I2C_Mem_Read parameters explained:
     * 1. &hi2c1: This is the name of the I2C that you're using. You set this in the CubeMX. Don't forget the '&'.
     * 2. IS4310_I2C_Chip_Address: The I2C address of the IS4310 (must be left-shifted).
     * 3. registerAdressToRead: The holding register address to read from the IS4310.
     * 4. I2C_MEMADD_SIZE_16BIT: You must indicate the memory addressing size. The IS4310 memory addressing is 16-bits.
     * This keyword is an internal constant of HAL libraries. Just write it.
     * 5. readResultArray: An 8-bit array where the HAL stores the read data.
     * 6. 2: The number of bytes to read. Since one holding register is 16 bits, we need to read 2 bytes.
     * 7. 1000: Timeout in milliseconds. If the HAL fails to read within this time, it will skip the operation
     * to prevent the code from getting stuck.
     */
    HAL_I2C_Mem_Read(&hi2c1, IS4310_I2C_Chip_Address, registerAdressToRead, I2C_MEMADD_SIZE_16BIT, readResultArray, 2, 1000);

    // Combine two bytes into a 16-bit result:
```

```
    readResult = readResultArray[0];
    readResult = readResult << 8;
    readResult = readResult | readResultArray[1];

    return readResult;
}

void writeHoldingRegister(uint16_t registerAdressToWrite, uint16_t value) {
    uint8_t IS4310_I2C_Chip_Address;  // I2C address of IS4310 chip (7-bit).
    IS4310_I2C_Chip_Address = 0x11; // IS4310 I2C address is 0x11 (7-bit).
    // STM32 HAL expects 8-bit address, so shift left by 1:
    IS4310_I2C_Chip_Address = IS4310_I2C_Chip_Address << 1;

    // The HAL library to write I2C memories needs the data to be in a uint8_t array.
    // So, lets put our uint16_t data into a 2 registers uint8_t array.
    uint8_t writeValuesArray[2];
    writeValuesArray[0] = (uint8_t) (value >> 8);
    writeValuesArray[1] = (uint8_t) value;

    /*
     * This is the HAL function to write to an I2C memory device. To be simple and easy to use, the IS4310 is designed to operate as an I2C
memory.
     *
     * HAL_I2C_Mem_Write parameters explained:
     * 1. &hi2c1: This is the name of the I2C that you're using. You set this in the CubeMX. Don't forget the '&'.
     * 2. IS4310_I2C_Chip_Address: The I2C address of the IS4310 (must be left-shifted).
     * 3. registerAdressToWrite: The holding register address of the IS4310 we want to write to.
     * 4. I2C_MEMADD_SIZE_16BIT: You must indicate the memory addressing size. The IS4310 memory addressing is 16-bits.
     * This keyword is an internal constant of HAL libraries. Just write it.
     * 5. writeValuesArray: An 8-bit array where we store the data to be written by the HAL function.
     * 6. 2: The number of bytes to write. Since one holding register is 16 bits, we need to write 2 bytes.
     * 7. 1000: Timeout in milliseconds. If the HAL fails to write within this time, it will skip the operation
     * to prevent the code from getting stuck.
     */
    HAL_I2C_Mem_Write(&hi2c1, IS4310_I2C_Chip_Address, registerAdressToWrite, I2C_MEMADD_SIZE_16BIT, writeValuesArray, 2, 1000);
}

while (1) {
    // This will store the potentiometer value:
    uint16_t potentiometerValue;
    // This will store the read value of the Holding Registers 1, 2 and 3:
    uint16_t holdingRegister1;
    uint16_t holdingRegister2;
    uint16_t holdingRegister3;

    // Read Holding Registers 1, 2 and 3:
    holdingRegister1 = readHoldingRegister(1);
    holdingRegister2 = readHoldingRegister(2);
    holdingRegister3 = readHoldingRegister(3);
    // If the value of each read Holding register is different from 0,
    // let's turn on the corresponding LED:
    if (holdingRegister1 >= 1) {
        HAL_GPIO_WritePin(RGB_Red_GPIO_Port, RGB_Red_Pin, GPIO_PIN_SET);
    } else {
        HAL_GPIO_WritePin(RGB_Red_GPIO_Port, RGB_Red_Pin, GPIO_PIN_RESET);
    }

    if (holdingRegister2 >= 1) {
        HAL_GPIO_WritePin(RGB_Green_GPIO_Port, RGB_Green_Pin, GPIO_PIN_SET);
    } else {
        HAL_GPIO_WritePin(RGB_Green_GPIO_Port, RGB_Green_Pin, GPIO_PIN_RESET);
    }

    if (holdingRegister3 >= 1) {
        HAL_GPIO_WritePin(RGB_Blue_GPIO_Port, RGB_Blue_Pin, GPIO_PIN_SET);
    } else {
        HAL_GPIO_WritePin(RGB_Blue_GPIO_Port, RGB_Blue_Pin, GPIO_PIN_RESET);
    }

    /*
     * Read ADC value from potentiometer (0-4095),
     * and write it to Holding Register 0.
     */
    HAL_ADC_Start(&hadc1); // Start the HAL ADC
    HAL_ADC_PollForConversion(&hadc1, 400); // Perform an ADC read
    // Get the ADC value:
    potentiometerValue = HAL_ADC_GetValue(&hadc1);
    // Store the ADC value to the Holding Register 0:
    writeHoldingRegister(0, potentiometerValue);
    // Stop the HAL ADC
    HAL_ADC_Stop(&hadc1);
}
```

7.3. Raspberry Pi Example

Coding for the IS4310 requires no dedicated library, making it easy to maintain and port to new Raspberry Pi boards or other single board computers (SBC).

This Python script communicates with the IS4310 Modbus RTU chip via I2C using a Raspberry Pi.

It demonstrates:

1. How to read a push button (simulating a sensor) and store its state in Holding Register 0.
2. How to control an RGB LED (simulating an actuator) using PWM on GPIO pins 12, 13, and 19, based on values in Holding Registers 1, 2, and 3.

A value of 0 turns off the LEDs, and a value of 100 sets them to maximum brightness.

This example uses the Kappa4310Rasp Evaluation Board. Check the Kappa4310Ard product page for more information.

You can download the full Raspberry Pi Python project from the IS4310 product page.

```
# IS4310 Modbus Code Example for Raspberry Pi
# ----------------------------------------------------------
# This Python script communicates with the IS4310 Modbus RTU chip via I²C using a Raspberry
Pi.
# It demonstrates how to read a push button (simulating a sensor) and store its value in
Holding Register 0.
# It also controls an RGB LED (simulating an actuator) using PWM pins 12, 13, and 19, based on
the values in Holding Registers 1, 2, and 3.
# A value of 0 turns off the LEDs, and a value of 100 sets them to maximum brightness.
#
#  You can test this code using the **Kappa4310Rasp Evaluation Board**.
# Buy it at: [www.inacks.com/kappa4310rasp](https://www.inacks.com/kappa4310rasp)
# Download the IS4310 datasheet at: www.inacks.com/is4310

from smbus2 import SMBus, i2c_msg
import RPi.GPIO as GPIO
import time

I2C_BUS = 1  # I2C bus number on Raspberry Pi (usually 1)
DEVICE_ADDRESS = 0x11  # 7-bit I2C address of the IS4310 Modbus RTU chip
GPIO.setmode(GPIO.BCM)  # Use BCM pin numbering scheme

# Define GPIO pins for three LEDs and push button
led_pin1 = 12
led_pin2 = 13
led_pin3 = 19
push_button_pin = 26

# Setup push button pin as input with internal pull-down resistor enabled
GPIO.setup(push_button_pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

# Setup LED pins as outputs
GPIO.setup(led_pin1, GPIO.OUT)
GPIO.setup(led_pin2, GPIO.OUT)
GPIO.setup(led_pin3, GPIO.OUT)

# Initialize PWM on LED pins at 1 kHz frequency
pwm1 = GPIO.PWM(led_pin1, 1000)
pwm2 = GPIO.PWM(led_pin2, 1000)
pwm3 = GPIO.PWM(led_pin3, 1000)

# Start PWM with 0% duty cycle (LEDs off initially)
pwm1.start(0)
pwm2.start(0)
pwm3.start(0)

def write_register(register, data):
    """
    Write a 16-bit data value to a 16-bit register address on the I2C device.

    :param register: 16-bit register address (split into high and low bytes)
    :param data: 16-bit data to write (split into high and low bytes)
    """
    high_addr = (register >> 8) & 0xFF  # Extract high byte of register address
    low_addr = register & 0xFF          # Extract low byte of register address
    data_high = (data >> 8) & 0xFF      # Extract high byte of data
    data_low = data & 0xFF              # Extract low byte of data

    # Open I2C bus, send write message: [register high, register low, data high, data low]
    with SMBus(I2C_BUS) as bus:
        msg = i2c_msg.write(DEVICE_ADDRESS, [high_addr, low_addr, data_high, data_low])
        bus.i2c_rdwr(msg)

def read_register(start_register):
    """
    Read a 16-bit value from a 16-bit register address on the I2C device.

    :param start_register: 16-bit register address to read from
    :return: 16-bit integer value read (big-endian)
    """
    high_addr = (start_register >> 8) & 0xFF  # High byte of register address
    low_addr = start_register & 0xFF          # Low byte of register address

    with SMBus(I2C_BUS) as bus:
        # Write register address first to set internal pointer
        write_msg = i2c_msg.write(DEVICE_ADDRESS, [high_addr, low_addr])
        # Prepare to read 2 bytes from the device
        read_msg = i2c_msg.read(DEVICE_ADDRESS, 2)
        bus.i2c_rdwr(write_msg, read_msg)

        data = list(read_msg)  # Read bytes as list of ints
        # Combine high and low bytes into 16-bit integer (big-endian)
        value = (data[0] << 8) | data[1]
        return value

try:
    while True:
        # Read push button state (0 or 1)
        button_value = GPIO.input(push_button_pin)

        # Write button state to register 0 of the device
        write_register(0, button_value)

        # Read PWM values from registers 1, 2, and 3
        pwm_val1 = read_register(1)
        pwm_val2 = read_register(2)
        pwm_val3 = read_register(3)

        # Cap PWM values at max 100 to avoid invalid duty cycles
        if pwm_val1 > 100:
            pwm_val1 = 100
        if pwm_val2 > 100:
            pwm_val2 = 100
        if pwm_val3 > 100:
            pwm_val3 = 100

        # Calculate duty cycles by inverting the PWM value (100 - value)
        # abs() used to ensure positive duty cycle, just in case
        duty1 = abs(pwm_val1 - 100)
        duty2 = abs(pwm_val2 - 100)
        duty3 = abs(pwm_val3 - 100)

        # Print duty cycle values for debugging (tab-separated)
        print(f"{duty1}\t{duty2}\t{duty3}")

        # Update PWM duty cycles to control LED brightness
        pwm1.ChangeDutyCycle(duty1)
        pwm2.ChangeDutyCycle(duty2)
        pwm3.ChangeDutyCycle(duty3)

        # Small delay to avoid excessive CPU load
        time.sleep(0.05)

except KeyboardInterrupt:
    # Gracefully handle Ctrl+C exit
    print("Exiting...")

finally:
    # Stop all PWM signals and cleanup GPIO pins on exit
    pwm1.stop()
    pwm2.stop()
    pwm3.stop()
    GPIO.cleanup()
```

8. PC/Mac Tools

The following third-party software options are provided for reference only. These applications are not developed, maintained, or endorsed by INACKS. We do not guarantee their functionality, compatibility, or compliance with the Modbus standard. Users should evaluate and choose software based on their specific needs.

8.1. Modbus Tools

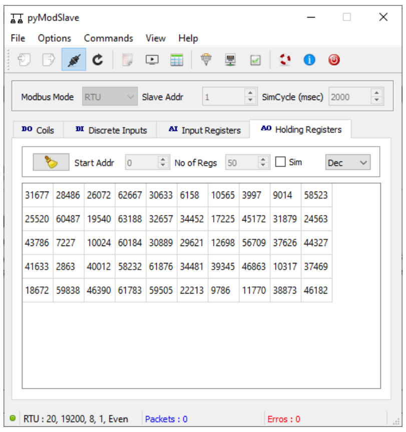
USB Isolated Modbus Device

Title: DamnModIsoUsb

Description:
The DamnModIsoUsb is a USB to Modbus RTU interface that allows a PC or Mac to operate as either a Modbus RTU Master or Slave. It includes a robust USB connector and galvanic isolation to ensure safe and reliable operation. The device is compatible with Python, pyModSlave, QModMaster, and other software or scripts that support serial communication.

Web:
www.inacks.com/damnmodisousb

Modbus Slave Software



Modbus Mode: RTU | Slave Addr: 1 | SimCycle (msec): 2000

Start Addr: 0 | No of Regs: 50 | Dec

31677	28486	26072	62667	30633	6158	10565	3997	9014	58523
25520	60487	19540	63188	32657	34452	17225	45172	31879	24563
43786	7227	10024	60184	30889	29621	12698	56709	37626	44327
41633	2863	40012	58232	61876	34481	39345	46863	10317	37469
18672	59838	46390	61783	59505	22213	9786	11770	38873	46182

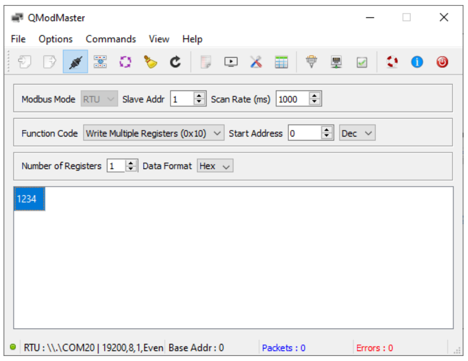
RTU : 20, 19200, 8, 1, Even | Packets : 0 | Erros : 0

Title: pyModSlave

Description:
pyModSlave is a free python-based implementation of a Modbus slave application for simulation purposes. You can install the python module or use the precompiled (for Windows only) stand alone GUI (Qt based) utility (unzip and run). pyModSlave also includes a bus monitor for examining all traffic on the bus.

Web:
https://sourceforge.net/projects/pymodslave/

Modbus Master Software



Modbus Mode: RTU | Slave Addr: 1 | Scan Rate (ms): 1000

Function Code: Write Multiple Registers (0x10) | Start Address: 0 | Dec

Number of Registers: 1 | Data Format: Hex

1234

RTU : \\.\COM20 | 19200,8,1,Even | Base Addr : 0 | Packets : 0 | Errors : 0

Title: QModMaster

Description:
QModMaster is a free Qt-based implementation of a Modbus master application. A graphical user interface allows easy communication with Modbus RTU and TCP slaves. QModMaster also includes a bus monitor for examining all traffic on the bus.

Web:
https://sourceforge.net/projects/qmodmaster/

Important:
Go to menu *Options* → *Settings*, and set the parameter *Base Addr* to 0 to avoid confusions with the Modbus Registers Addresses.

8.2. I2C Tools

USB I2C Master Device

Title: DamnI2cUsb

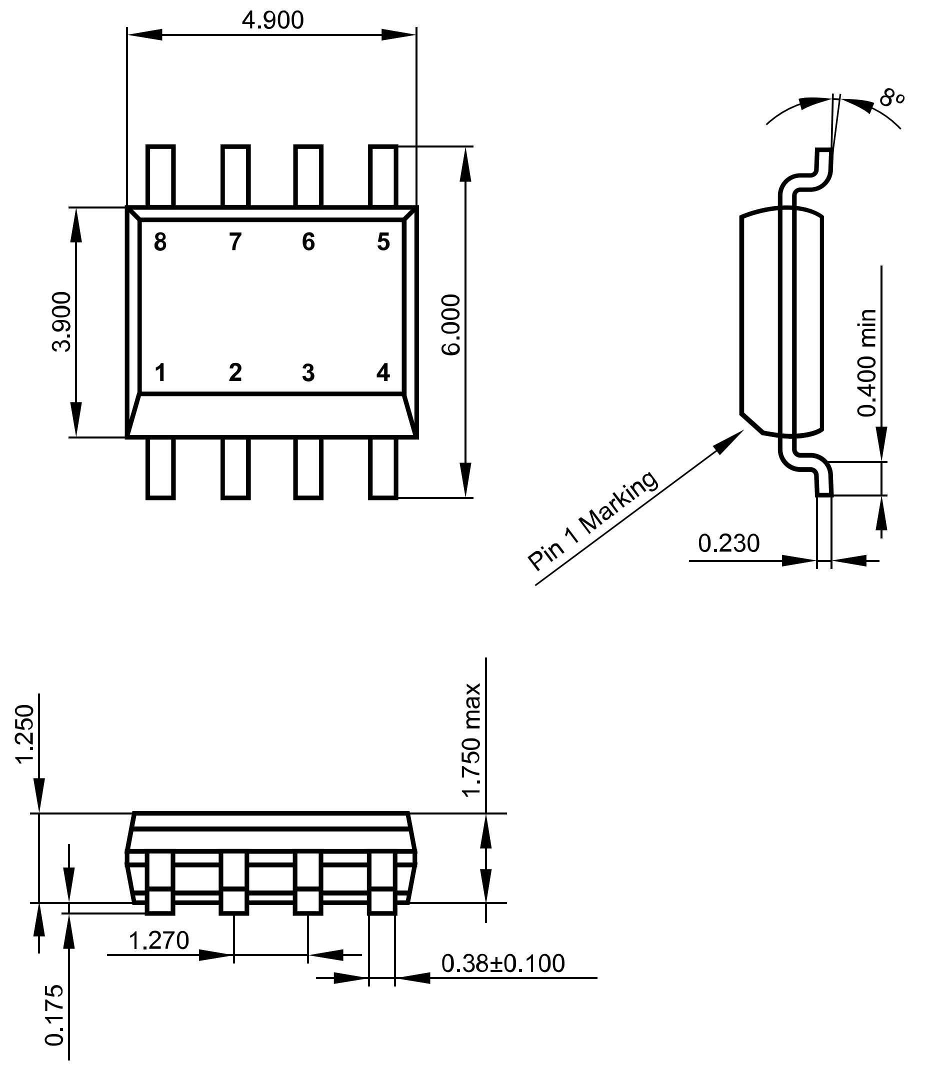
Description:
The DamnI2cUsb is a USB to I2C-Master interface that allows a PC or Mac to operate as an I2C Master Device, similar to a microcontroller.

Communication is handled over a serial link, making it compatible with Python and other scripting or programming languages. The tool allows reading from and writing to the IS4320 memory map, making it especially useful during the debugging stage.

Web:
www.inacks.com/damni2cusb

9. Mechanical

SO8N Package

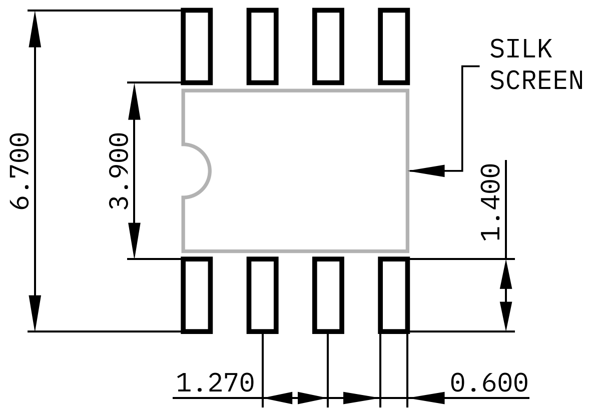


Units: millimeters

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

Date	Revision Code	Description
September 2025	ISDOC125**D**	- Updated “Mechanical” section. - Updated “Electrical Specification” section.
June 2025	ISDOC125**C**	- Added “Firmware Implementation Guide” section. - Updated pictures from “Product Selection Guide” section. - Typo in the MBBDR Register section: “Name” field was incorrectly written as “MBADD” and has been corrected to “MBBDR”. - Added “Modbus Software Tools” section.
February 2025	ISDOC125**B**	- Added “Cable Wiring” section.
February 2025	ISDOC125**A**	- 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