IS4310-ISO485M6

Isolated RS485 Modbus RTU Slave Module with RJ45

Presentation

The IS4310-ISO485M6 is a ready-to-operate module integrating the Modbus RTU Slave stack chip IS4310 with an isolated RS485 Transceiver. This solution reduces to the minimum expression the design effort of integrating a Modbus RTU Slave with an isolated RS485 electrical interface.

The module is daisy-chained: it integrates two RJ45 connectors with the standard Modbus pinout.

The module can be directly soldered to your PCB with its castellated holes, or you can solder a pin header and use it as a module.

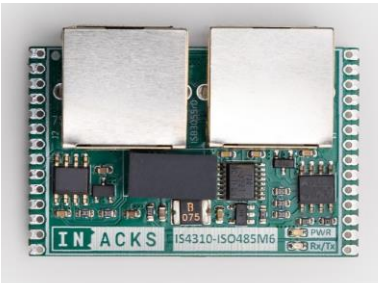
Isolation Benefits

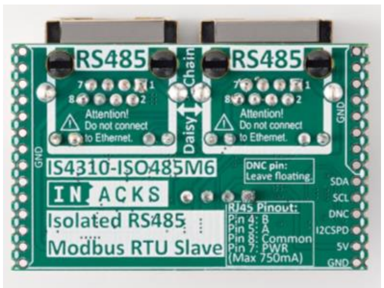
Isolation is a good solution for industrial environments with long-distance cables, significant ground potential differences between devices, or high electrical noise.

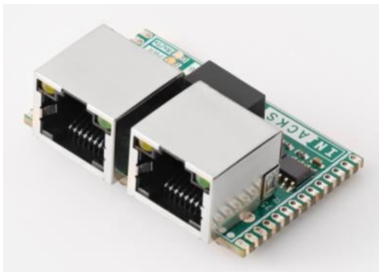
Module Characteristics

Electrical Characteristics	
Module Voltage	5V
I2C Compatible Voltage Levels	3.3V, 5V
Isolated RS485	Yes

Modbus Characteristics	
Supported Function Codes:	3 (0x03) - Read Holding Registers 6 (0x06) - Write Single Register 16 (0x10) - Write Multiple Registers
Holding Registers:	500
Operating Mode:	RTU
Electrical Interface:	Isolated RS485



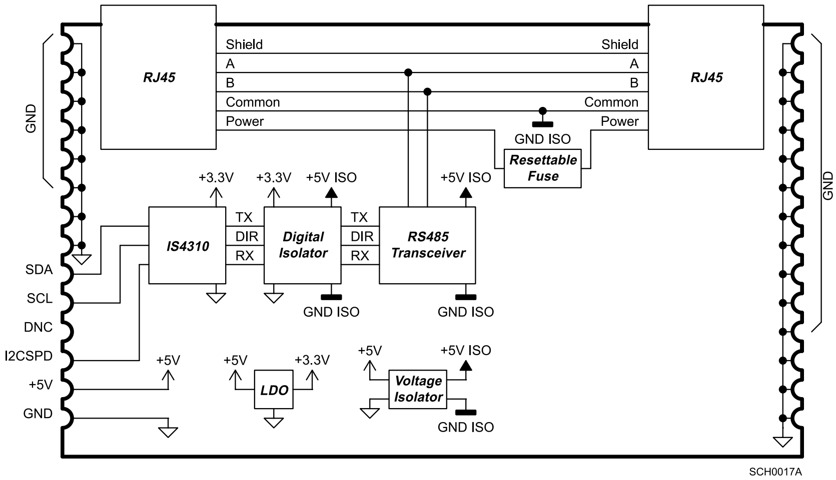




Product Selection Guide

	Part Number		Form Factor	Physical Layer	Stack	Description
Only Stack	IS4310-S8		SO8N	UART 3.3V	Modbus RTU Server	Modbus RTU Slave Stack Chip. View Product
Stack with Physical Layer	IS4310-485M2		Castellated Holes Module	RS485	Modbus RTU Server	IS4310 with RS485 Transceiver. Industrial communications. View Product
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. View Product
Stack with Physical Layer	IS4310-232M4		Castellated Holes Module	RS232	Modbus RTU Server	IS4310 with RS232 Transceiver. View Product
Evaluation Boards	Kappa4310Ard		Arduino Compatible	RS485	Modbus RTU Server	IS4310 Evaluation Board with RS485 Transceiver. Compatible with Arduino. View Product
Evaluation Boards	Kappa4310Rasp		Raspberry Pi Compatible	RS485	Modbus RTU Server	IS4310 Evaluation Board with RS485 Transceiver. Compatible with Raspberry Pi. View Product

1. Description



The IS4310-ISO485M6 is a compact (44 × 28 mm) module with castellated holes, designed for PCB mounting to function as an RS485 Modbus RTU Slave. It features two key components: the Modbus RTU Slave Stack (IS4310) and an RS485 transceiver.

The module features two RJ45 daisy-chain connectors and supports optional Bus Power daisy-chaining. When using a powered bus, the maximum allowable daisy-chained current is 750mA. A resettable fuse protects the module from excessive currents. Additionally, when using a Powered Modbus Serial Line, ensure the maximum cable current rating is not exceeded.

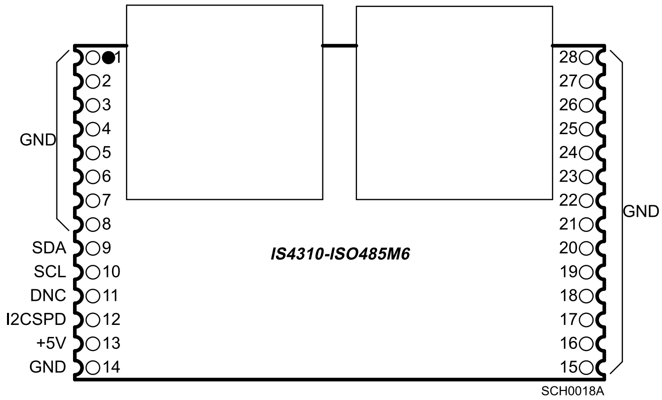
The return path for the PWR pad is through the GND pads, which are connected to the Common signal of the Serial Line.

The Shield pin is daisy-chained and not connected to any pad. All cable shields should be connected to Common and Protective Ground at a single point for the entire bus, ideally at the master device. Ensure that you use shielded cables and connectors to maintain proper electrical continuity across all cable shields on the bus.

The module operates at a fixed voltage of 5V. The I2C pads (SDA and SCL) are open-drain and compatible with both 3.3V and 5V. The DNC pad must be left unconnected. The I2CSPD pin is used for I2C speed selection. The module includes multiple GND pads, and proper soldering of all pads is essential for mechanical stability and durability when attaching the module to the main PCB.

The module also features a green LED to indicate power status and a yellow LED to signal data transmission or reception. Each RJ45 connector includes the same two LEDs with identical functionality.

1.1. Module Pinout



Pad	Name	Type	Description
9	SDA	Open Drain 3.3V (5V Tolerant)	SDA pin of the IS4310: Open drain, it requires pull-up.
10	SCL	Open Drain 3.3V (5V Tolerant)	SCL pin of the IS4310: Open drain, it requires pull-up.
11	DNC	Do Not Connect	This pad must be left floating.
12	I2CSPD	Analog Input 0 to 3.3V	I2CSPD pin of the IS4310: I2C-Serial Interface Speed Selection. • For 100kHz pull to GND. • For 400kHz make a voltage divider of VDD/2 (1.65V). • For 1MHz pull to 3.3V. Attention: Voltage in this pin above 4V will damage the IS4310.
13	+5V	Module Power (Power In)	Power supply for the module.
1-8, 14-28	GND	Ground	Ground reference pad. GND pads are connected to the “Common” of the RS485 bus. GND pads are not connected to the shield of the connector and cable of the RS485 bus. GND pads are the return of the optional Power on the bus (PWR Pad).

SCL and SDA Pads

I2C-Compatible Bus Interface Pads.

Both pads 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 2.2kΩ for Fast Mode (400kbps) at both 3.3V and 5V.

+5V Pad

Module Power Supply Pad.

This pad is the power input for the entire module. The module includes an LDO to regulate the voltage down to 3.3V for the IS4310 power supply.

5V must be supplied to this pad. Bypass capacitors are included on the module, no need to place them outside of the module.

GND Pads

Module Ground.

The GND pads the 0V of the module.

These pads are also connected to the Common signal of the RS485 bus and therefore serve as the return path for the PWR Pad.

The common signal of the RS485 bus must be connected to protective ground, preferably at a single point. The recommended location is at the master device.

GND pads are not connected to the shield of the RS485 cables and connectors, as all shields should be connected to protective ground at a single point for the entire bus, ideally at the master device.

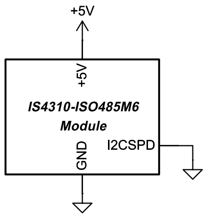
The module has multiple GND pads, all of which must be soldered to ensure proper mechanical attachment of the module to the main PCB. This is especially important when the RJ45 connectors are plugged in, as a significant amount of force is applied to the module. Failing to solder all pads or poor soldering can cause the module to detach from the main PCB.

I2CSPD Pad

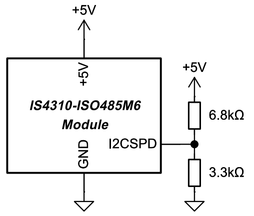
I2C-Serial Interface Speed Selection Pad.

This pad is directly connected to the pin I2CSPD of the IS4310. It configures the IS4310 internal I2C-Serial Interface timings and filters to properly work with the selected bus speed.

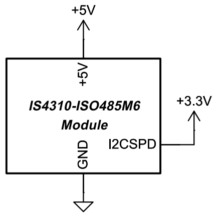
- For a **100kHz** setting, set the I2CSPD pad to GND.



- For a **400kHz** setting, set the I2CSPD to 1.65V using a voltage divider. This can be achieved using a 6.8kΩ and a 3.3kΩ resistors from the I2CSPD pad: one to 5V and the other to GND. Please note that applying a voltage higher than 3.3V to I2CSPD pad would permanently damage the IS4310 IC.



- For a **1000MHz** setting, set the I2CSPD pad to 3.3V. Please note that pulling I2CSPD to 5V would permanently damage the IS4310 IC.



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

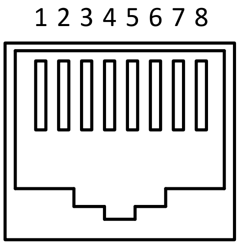
Ensure a proper match between the actual operating speed and the configured speed at the I2CSPD pad: If your bus works at 100kHz, ensure the I2CSPD pad is tied to VSS. If it works at 400kHz ensure the pad is at 1.65V. If it works at 1000MHz, ensure the pad is at 3.3V.

1.2. RJ45 Connectors

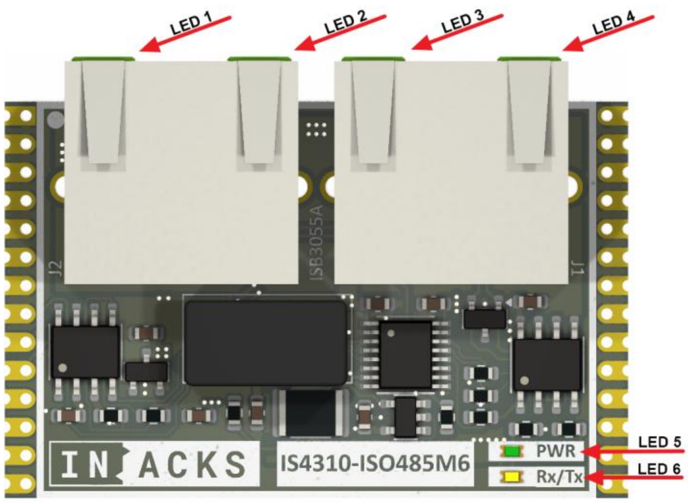
Typical Modbus Serial Line connectors include Screw Terminals, RJ45, and D-Sub 9-pin (commonly known as DB9), among others. The device-side connector must be female, while the cable-side connector must be male.

When selecting a RJ45 cable, ensure it has shield and make sure to connect the cable shield to the connector shield to ensure proper electrical continuity across all cable shields on the bus. Additionally, when using a Powered Modbus Serial Line, ensure the maximum cable current rating is not exceeded.

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.

<table>
<tr><th colspan="2">RJ45 Connector for RS485 Modbus</th></tr>
<tr><td rowspan="9">
</td><td>1: NC</td></tr>
<tr><td>2: NC</td></tr>
<tr><td>3: NC</td></tr>
<tr><td>4: B (D1)</td></tr>
<tr><td>5: A (D0)</td></tr>
<tr><td>6: NC</td></tr>
<tr><td>7: Bus Power Supply (optional)</td></tr>
<tr><td>8: Common</td></tr>
<tr><td>Shield: Cable Shield</td></tr>
<tr><td colspan="2">Attention!
The RJ45 connector is intended for the Modbus RS485 bus and must not be connected to an Ethernet network. Connecting it to an Ethernet network may cause damage to Ethernet devices or this device.</td></tr>
</table>

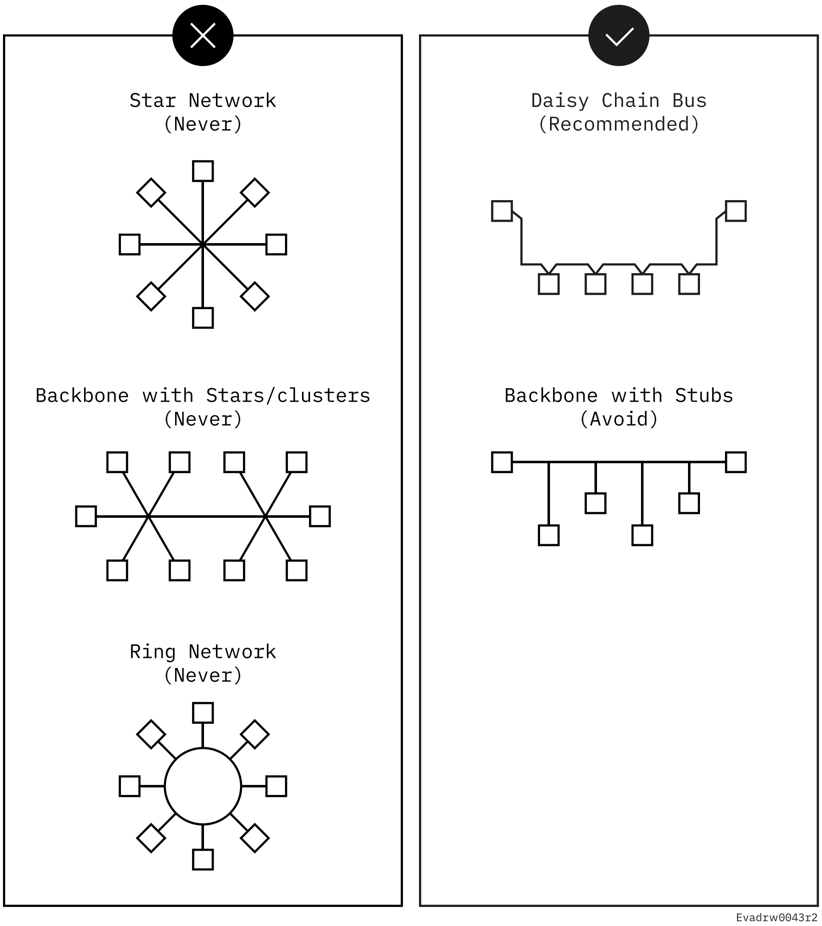
1.3. LEDs

<table>
<tr><th>LED Location</th><th>LED #</th><th>Description</th></tr>
<tr><td rowspan="2">
</td><td>1, 3, 5</td><td>Power Indicator
- <u>Green</u>: Module On
- <u>Off</u>: Module Off</td></tr>
<tr><td>2, 4, 6</td><td>Data Indicator
- <u>Yellow Flashing</u>: Transmitting or Receiving Data
- <u>Off</u>: No Data</td></tr>
</table>

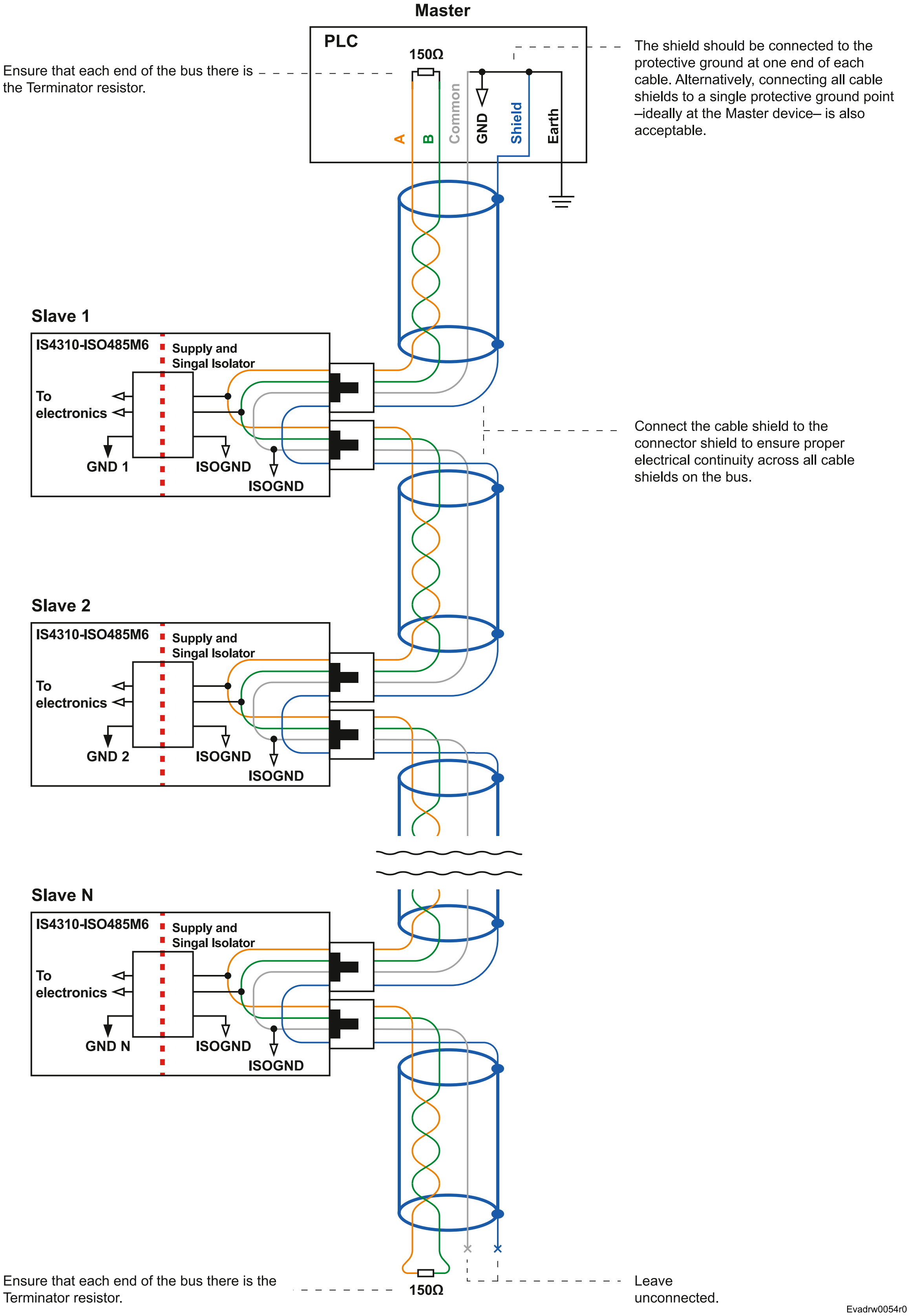
2. Bus Recommendations

2.1. Topology

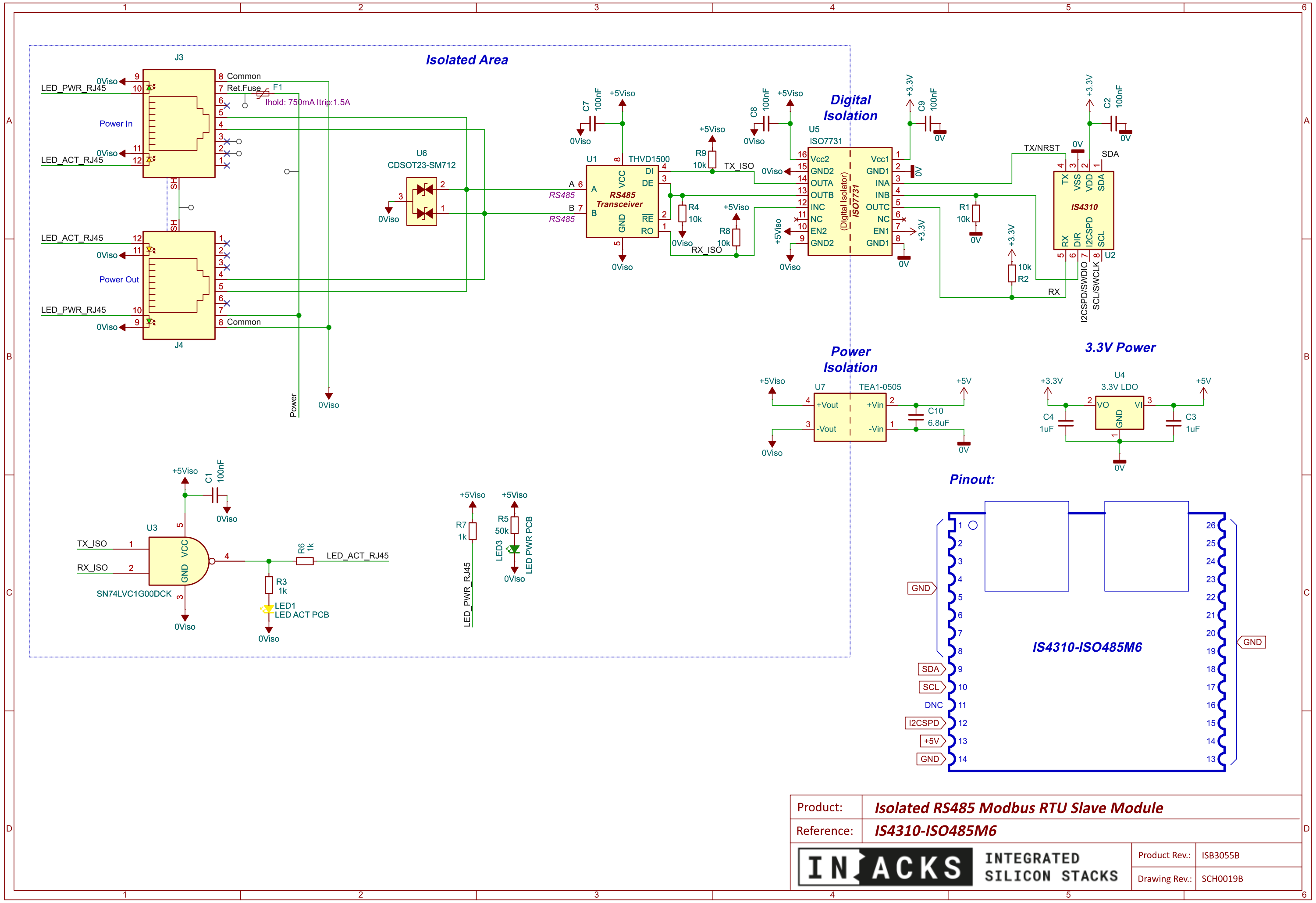
In an RS485 setup without a repeater, a single trunk cable runs through the system, with devices connected in a daisy-chain manner. Short cables derivations (stubs) are also allowed but not recommended. Keep the derivation distance as short as possible. Other topologies are not allowed.



2.2. Cable Wiring



3. Schematic


Isolated Area
J3
Power In
J4
Power Out
LED_PWR_RJ45
LED_ACT_RJ45
Common
Ret.Fuse
F1
Ihold: 750mA Itrip:1.5A
U6
CDSOT23-SM712
U1
THVD1500
RS485 Transceiver
Digital Isolation
U5
ISO7731
U2
IS4310
TX/NRST
SDA
TX_ISO
RX_ISO
Power Isolation
U7
TEA1-0505
3.3V Power
U4
3.3V LDO
U3
SN74LVC1G00DCK
LED1
LED ACT PCB
LED3
LED PWR PCB
Pinout:
IS4310-ISO485M6
GND
SDA
SCL
DNC
I2CSPD
+5V
Product:
Isolated RS485 Modbus RTU Slave Module
Reference:
IS4310-ISO485M6
INACKS INTEGRATED SILICON STACKS
Product Rev.:
ISB3055B
Drawing Rev.:
SCH0019B

4. 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.

4.1. 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);
}
```

4.2. 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);
}
```

4.3. 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.

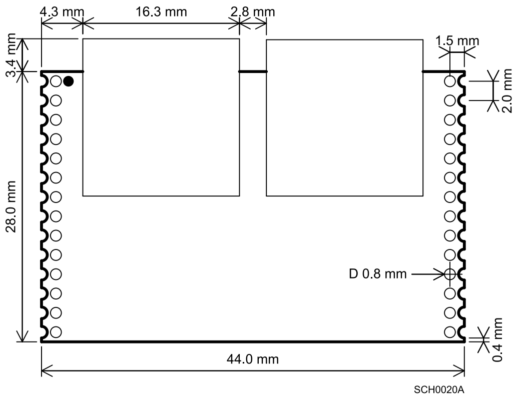
```
# IS4310 Modbus Code Example for Raspberry Pi
# ----------------------------------------------------------
# This Python script communicates with the IS4310 Modbus RTU chip via I²C using a Raspberry
Pi.
# It demonstrates how to read a push button (simulating a sensor) and store its value in
Holding 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)
    """
```

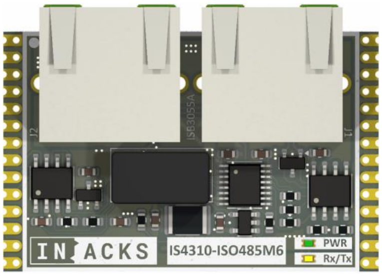
```
    high_addr = (register >> 8) & 0xFF  # Extract high byte of register address
    low_addr = register & 0xFF          # Extract low byte of register address
    data_high = (data >> 8) & 0xFF      # Extract high byte of data
    data_low = data & 0xFF              # Extract low byte of data

    # 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()
```

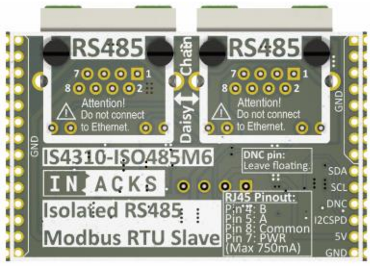
```
    pwm2.stop()
    pwm3.stop()
    GPIO.cleanup()
```

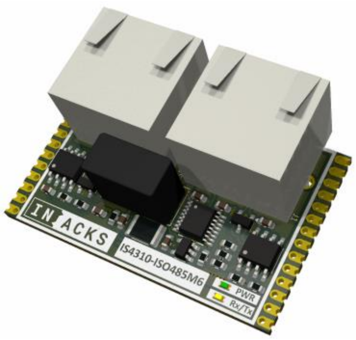
5. Mechanical Dimensions

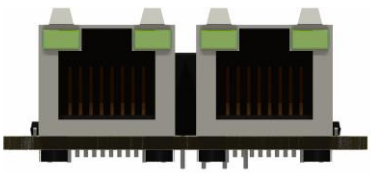
M6 Package







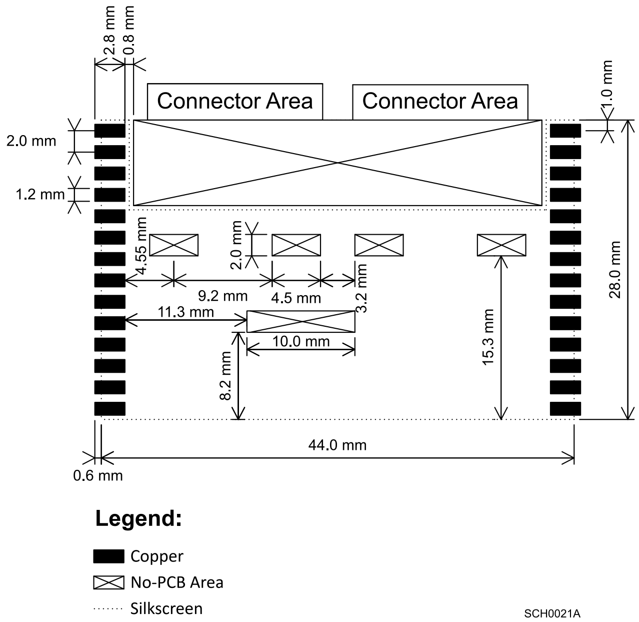


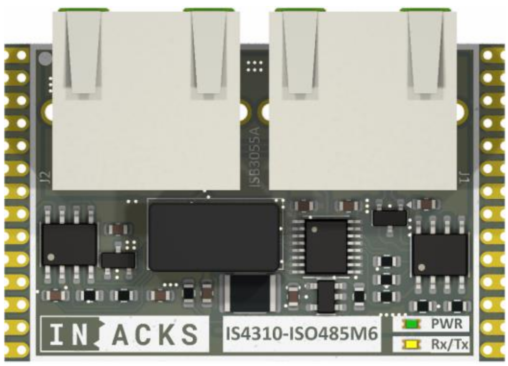


Units
Millimeters

Notes
This drawing is for general information only.
Not to scale.

M6 Recommended Footprint





Units
Millimeters

Notes
This drawing is for general information only.
Not to scale.

Content

Appendix

Revision History

Document Revision

Date	Revision Code	Description
June 2025	ISDOC127**D**	Changed renders for pictures. Added Arduino, STM32 and Raspberry Pi examples.
June 2025	ISDOC127**C**	Schematic revision.
February 2025	ISDOC127**B**	Added “Cable Wiring” diagram.
February 2025	ISDOC127**A**	Initial Release.

Module Revision

Date	Revision Code	Description
June 2025	ISB3055**B**	Layout revision.
February 2025	ISB3055**A**	Initial Release.

Documentation Feedback

Feedback and error reporting on this document are very much appreciated.

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.