ROWLAND TECHNOLOGY

Interface Board

Product Datasheet & Developer Guide

Key Features

ADC • DAC • I2C • GPIO • Servo • SPI • UART

USB & WiFi Connectivity • 3.3V / 5V Logic

Supports Python, C#, and Flowcode

Table of Contents

1. Product Overview

The Rowland Technology Interface Board is a compact hardware device that bridges PC-based software with embedded electronic signals at 3.3 V or 5 V logic levels. It allows developers to prototype, test, and deploy embedded-style peripherals directly from a Windows or Linux PC without writing any microcontroller firmware.

The board acts as a SCADA-style slave device: your host application sends commands over USB or WiFi, and the Interface Board handles the low-level hardware signalling on your behalf. This makes it ideal for rapid prototyping, lab automation, robotics, and IoT gateway projects.

1.1 Supported Interfaces

Interface	Description
ADC	Analogue-to-Digital Converter — reads analogue voltage levels as 8-bit or 10-bit values
DAC	Digital-to-Analogue Converter — outputs a 5-bit analogue voltage
GPIO	General Purpose I/O — read input pins or drive output pins high/low
I2C	Two-wire serial bus — communicate with sensors, displays, EEPROMs, and other I2C devices
SPI	Four-wire serial bus — high-speed full-duplex communication with SPI peripherals
UART	Serial data link — transmit and receive bytes at configurable baud rates (USB firmware only)
Servo	Servo motor control — drive RC servo channels with 8-bit or 16-bit position resolution
PWM	Pulse-Width Modulation — variable duty-cycle signal output with configurable prescaler

1.2 Connectivity Options

USB Firmware

The USB variant connects directly to a host PC via USB. It presents as a virtual COM port (serial over USB), which means no specialist drivers are required beyond the included USB driver. Compatible with Windows and most other USB host devices.

WiFi Firmware

The WiFi variant connects the board to your local network over 802.11 WiFi. Once configured, any device on the network (or the internet, with appropriate routing) can control the board. This enables remote-control and distributed applications.

NOTE: *UART slave functionality is not available when using the WiFi firmware variant.*

1.3 Logic Level

All I/O signals on the Interface Board operate at either 3.3 V or 5 V logic levels depending on the board variant supplied. Ensure external devices are compatible with the board's logic level to avoid damage.

1.4 Programming Language Support

Example code and library files are provided for:

- Python (RT_IB.py library)
- C# (example projects included)
- Flowcode (component file included)

2. Getting Started

2.1 What You Need

- Interface Board (USB or WiFi variant)
- USB cable (for power and/or USB communication)
- Windows or Linux host PC
- Python 3.x, C# development environment, or Flowcode (as required)
- pyserial Python package (pip install pyserial)
- USB Driver (included in the repository under the USB Driver folder)

2.2 USB Setup

Follow these steps to get the USB variant running:

- Install the USB driver from the USB Driver folder in the repository.
- Connect the Interface Board to your PC via USB. Windows will assign a COM port number.
- Open Device Manager and note the COM port number assigned (e.g. COM3).
- Update the comport variable in your example program to match.
- Run an example program from the Example Programs folder to verify communication.

2.3 WiFi Setup

The WiFi variant must be configured once before it can join your network:

- Power the board via USB or a 5V supply.
- On your PC or phone, connect to the WiFi network named "InterfaceBoard" (password is "interfaceboard" in lowercase).
- Open a browser and navigate to 192.168.4.1
- Enter your network SSID and password in the configuration portal and save.
- The "InterfaceBoard" access point will disappear and the board will connect to your network, obtaining a local IP address from your router.
- The credentials are stored in non-volatile memory. If the network becomes unavailable, the board reverts to setup mode automatically.

NOTE: *Find the board's IP address by checking your router's connected device list or using a network scanner tool such as nmap or Advanced IP Scanner.*

2.4 Updating Firmware

Firmware can be changed using a Microchip PICkit 3, PICkit 4, or similar compatible programmer. Contact Rowland Technology for available firmware variants.

2.5 Installing the Python Library

Copy the RT_IB.py library file from the Example Programs/Python folder into the same directory as your Python script before importing it:

```
import RT_IB
ib = RT_IB.Create()
```

Also ensure the pyserial package is installed:

```
pip install pyserial
```

3. Python API Reference

The RT_IB Python library provides the Create class, which encapsulates all communication with the Interface Board over a serial connection. All method calls send a binary command frame over the serial port and read back the response.

3.1 Connection Methods

Method	Parameters	Returns	Description
`ComOpen(port)`	`port: int — COM number`	`None`	Opens the serial connection to the board at 115200 baud, 8N1, 1s timeout.
`ComClose()`	`None`	`None`	Closes the serial connection. Always call this when finished.

NOTE: *Always call ComClose() at the end of your program to release the serial port.*

3.2 ADC Methods

Read analogue voltage levels from the board's analogue input pins.

Method	Parameters	Returns	Description
`ADCSample8(channel)`	`channel: int`	`int (0–255)`	Reads an analogue value from the specified channel as an 8-bit value.
`ADCSample10(channel)`	`channel: int`	`int (0–1023)`	Reads an analogue value from the specified channel as a 10-bit value.

3.3 DAC Methods

Control the analogue output voltage.

Method	Parameters	Returns	Description
`DACEnable()`	`None`	`None`	Enables the DAC output.
`DACDisable()`	`None`	`None`	Disables the DAC output.
`DACOutput(value)`	`value: int (0–31)`	`None`	Writes a 5-bit value (0–31) to the DAC output.

3.4 GPIO Methods

Read and write digital I/O pins.

Method	Parameters	Returns	Description
`IOSetOutputPin(pin, state)`	`pin: int, state: int (0 or 1)`	`None`	Drives the specified output pin high (1) or low (0).
`IOGetInputPin(pin)`	`pin: int`	`int (0 or 1)`	Reads the logic level of the specified input pin.

3.5 I2C Methods

Communicate with I2C devices. Always call I2CInitialise() before using I2C.

Method	Parameters	Returns	Description
`I2CInitialise()`	`None`	`None`	Enables the I2C interface. Must be called before any I2C transaction.
`I2CStart()`	`None`	`None`	Sends an I2C START condition.
`I2CStop()`	`None`	`None`	Sends an I2C STOP condition.
`I2CRestart()`	`None`	`None`	Sends an I2C repeated START condition.
`I2CSend(data)`	`data: int (0–255)`	`int (ACK status)`	Sends one byte to the I2C bus. Returns 0 on ACK, non-zero on NACK.
`I2CReceive(last)`	`last: int (0 or 1)`	`int (0–255)`	Reads one byte from the I2C bus. Pass last=1 for the final byte to send NACK.

NOTE: *I2C device addresses must be shifted left by 1 bit. To address device 0x40 in write mode: 0x40 << 1. Read mode: (0x40 << 1) | 0x01.*

3.6 SPI Methods

Communicate with SPI devices. Always call SPIInitialise() before using SPI.

Method	Parameters	Returns	Description
`SPIInitialise()`	`None`	`None`	Enables the SPI interface. Must be called before any SPI transaction.
`SPIPrescaler(scaler)`	`scaler: int`	`None`	Sets the SPI clock prescaler to adjust the bus speed.
`SPITransfer(data)`	`data: int (0–255)`	`int (0–255)`	Sends one byte and simultaneously reads one byte (full-duplex). Returns the received byte.

3.7 UART Methods

Use the board as a UART bridge. Available on USB firmware only.

Method	Parameters	Returns	Description
`UARTInitialise()`	`None`	`None`	Enables the UART interface.
`UARTBaud(rate)`	`rate: int`	`None`	Sets the baud rate. Consult firmware documentation for rate codes.
`UARTTransmit(data)`	`data: int (0–255)`	`None`	Writes one byte to the UART transmit buffer.
`UARTReceive()`	`None`	`int (0–255)`	Reads one byte from the UART receive buffer.
`UARTReceiveCount()`	`None`	`int`	Returns the number of bytes currently waiting in the receive buffer.

3.8 Servo Methods

Control RC servo motors connected to the board's servo output channels.

Method	Parameters	Returns	Description
`ServoEnable(channel)`	`channel: int`	`None`	Enables the specified servo output channel.
`ServoDisable(channel)`	`channel: int`	`None`	Disables the specified servo output channel.
`ServoSetPosition8(channel, position)`	`channel: int, position: int (0-255)`	`None`	Sets servo position as an 8-bit value.
`ServoSetPosition16(channel, position)`	`channel: int, position: int (0-65535)`	`None`	Sets servo position as a 16-bit value for finer control.

3.9 PWM Methods

Generate PWM signals for motor speed control, dimming, or other applications.

Method	Parameters	Returns	Description
`PWMEnable(channel)`	`channel: int`	`None`	Enables the specified PWM output channel.
`PWMDisable(channel)`	`channel: int`	`None`	Disables the specified PWM output channel.
`PWMSetPrescaler(scaler)`	`scaler: int`	`None`	Sets the PWM prescaler, adjusting the output frequency for all channels.
`PWMSetDuty8(channel, value)`	`channel: int, value: int (0–255)`	`None`	Sets the PWM duty cycle as an 8-bit value.
`PWMSetDuty10(channel, value)`	`channel: int, value: int (0-1023)`	`None`	Sets the PWM duty cycle as a 10-bit value for finer resolution.

4. Usage Examples (Python)

4.1 Blink an LED

This example toggles a GPIO output pin on and off 10 times to blink an LED:

```
import time
import RT_IB

ib = RT_IB.Create()
ib.ComOpen(3)           # Replace 3 with your actual COM port number

for _ in range(10):
    ib.IOSetOutputPin(0, 1)  # Pin 0 HIGH
    time.sleep(0.5)
    ib.IOSetOutputPin(0, 0)  # Pin 0 LOW
    time.sleep(0.5)

ib.ComClose()
```

4.2 Read ADC and Control PWM

This example reads an analogue input and maps it to a PWM duty cycle — useful for dimming an LED based on a potentiometer or light sensor:

```
import time
import RT_IB

ib = RT_IB.Create()
ib.ComOpen(3)           # Replace 3 with your actual COM port number
ib.PWMEnable(0)

for _ in range(20):
    sample = ib.ADCSample8(0)   # Read channel 0 (0-255)
    ib.PWMSetDuty8(0, sample)   # Map directly to PWM duty
    time.sleep(0.5)

ib.PWMDisable(0)
ib.ComClose()
```

4.3 I2C Sensor Read/Write

This example demonstrates a complete I2C transaction: writing configuration bytes to a device at address 0x40 and then reading back two bytes of data:

```
import RT_IB

ib = RT_IB.Create()
ib.ComOpen(3)
ib.I2CInitialise()

# --- Write two bytes to device 0x40, register 0x00 ---
ib.I2CStart()
ib.I2CSend(0x40 << 1)       # Address + write bit
```

```
ib.I2CSend(0x00)              # Internal register address
ib.I2CSend(0xAB)              # Data byte 0
ib.I2CSend(0xCD)              # Data byte 1
ib.I2CStop()

# --- Read two bytes back ---
ib.I2CStart()
ib.I2CSend(0x40 << 1 | 0x01) # Address + read bit
ib.I2CSend(0x00)               # Internal register to read
ib.I2CRestart()
byte0 = ib.I2CReceive(0)      # Not last byte
byte1 = ib.I2CReceive(1)      # Last byte — sends NACK
ib.I2CStop()

print(f'Received: {byte0:#04x}, {byte1:#04x}')
ib.ComClose()
```

4.4 Servo Position Control

This example sweeps a servo from one end of its range to the other:

```
import time
import RT_IB

ib = RT_IB.Create()
ib.ComOpen(3)
ib.ServoEnable(0)

# Sweep from 0 to 255
for pos in range(0, 256, 5):
    ib.ServoSetPosition8(0, pos)
    time.sleep(0.05)

# Sweep back from 255 to 0
for pos in range(255, -1, -5):
    ib.ServoSetPosition8(0, pos)
    time.sleep(0.05)

ib.ServoDisable(0)
ib.ComClose()
```

4.5 DAC Output Ramp

This example ramps the DAC output through all 32 levels and then disables it:

```
import time
import RT_IB

ib = RT_IB.Create()
ib.ComOpen(3)
ib.DACEnable()

for level in range(32):
```

```
    ib.DACOutput(level)
    time.sleep(0.1)

ib.DACDisable()
ib.ComClose()
```

5. Deployment Guide

5.1 Bench / Prototyping

For a desktop bench setup:

- Connect the board via USB to your development PC.
- Install the USB driver from the repository.
- Use the Python examples to interactively test peripherals.
- Open Device Manager to confirm the COM port and update your script accordingly.
- Power consumption is via the USB bus — no external power supply needed in most cases.

5.2 Standalone PC Application

To embed the Interface Board in a desktop application (e.g. a C# or Python SCADA tool):

- Bundle the RT_IB.py library or the C# examples with your application.
- Hard-code or store the COM port in a configuration file — allow it to be user-configurable.
- Open the COM port at application startup and close it on exit.
- Handle serial exceptions: the board may disconnect if the USB cable is removed.
- Use threading or async patterns if the UI must remain responsive during long operations.

5.3 WiFi / Networked Deployment

For remote or networked installations:

- Configure the WiFi credentials on-site before deploying the device.
- Assign a static IP address to the board via your router's DHCP reservation feature for predictable addressing.
- Document the board's IP address and use it in place of a COM port in networked variants of the library.
- If deploying over the internet, use a VPN or SSH tunnel rather than exposing the device directly — the board has no built-in authentication.
- Monitor connectivity: if the WiFi network drops, the board reverts to setup mode and will need reconfiguring.

5.4 Linux Deployment

The Python library supports Linux. Note the following differences:

- On Linux the COM port format is /dev/rfcomm{N} rather than a Windows COM port number.
- A known issue exists where ComOpen may address the wrong device on Linux — verify the correct device node before use.
- Ensure the user running the script has permission to access the serial device (add to the dialout group: sudo usermod -aG dialout $USER).

5.5 Flowcode Integration

The repository includes a Flowcode component for use with the Matrix Multimedia Flowcode development environment. This allows graphical programming of the Interface Board without

writing code manually. Import the component from the Flowcode Component folder in the repository.

5.6 Best Practices

- Always close the COM port (ComClose()) before exiting your application to prevent port lock-up.
- Avoid opening the same COM port from two programs simultaneously.
- Add error handling around ComOpen() — the port may not be available if the board is disconnected.
- Use a short delay (time.sleep) between rapid consecutive API calls to allow the board to process each command.
- For production use, log all serial errors to aid debugging.

6. Troubleshooting

Problem	Solution
Board not detected in Device Manager	Install the USB driver from the repository's USB Driver folder. Try a different USB cable or port.
ComOpen() throws an exception	Check the COM port number in Device Manager. Ensure no other application (e.g. a terminal emulator) is already using the port.
Wrong device addressed on Linux	Known issue. Manually specify the full device path (e.g. /dev/ttyUSB0) instead of using ComOpen() with a number.
WiFi access point does not appear	Ensure the board is powered. Try power-cycling it. Confirm you have the WiFi firmware variant.
Cannot reach 192.168.4.1	Ensure your device is connected to the "InterfaceBoard" WiFi network, not your normal network. Disable mobile data on phones.
Board not connecting to home WiFi	Double-check the SSID and password (case-sensitive). If using a 5 GHz network, the board may only support 2.4 GHz.
API calls return -1 or unexpected values	Confirm the baud rate is 115200 and timeout is set. Ensure the board is powered and not in setup mode.
UART not working	UART is unavailable on the WiFi firmware variant. Switch to the USB firmware variant if UART is required.

7. Technical Reference

7.1 Serial Protocol

The Interface Board communicates over a serial connection at 115200 baud, 8 data bits, no parity, 1 stop bit (8N1), with a 1-second read timeout. Commands are sent as binary byte arrays and responses are binary byte arrays of fixed length.

Parameter	Value
Baud Rate	115200
Data Bits	8
Parity	None
Stop Bits	1
Timeout	1 second
Logic Level	3.3V or 5V (variant dependent)
Firmware Languages	C (Microchip PIC firmware)
Host Languages	Python, C#, Flowcode
License	GPL-3.0

7.2 Command Opcodes

The following opcode table is derived from the RT_IB.py library source code:

Opcode	Method	Notes
`0x80`	`IOSetOutputPin`	pin, state
`0x81`	`IOGetInputPin`	pin; returns pin state
`0x82`	`SPIInitialise`	no parameters
`0x83`	`SPITransfer`	data; returns received byte
`0x85`	`SPIPrescaler`	scaler
`0x86`	`I2CInitialise`	no parameters
`0x87`	`I2CStart`	no parameters
`0x88`	`I2CRestart`	no parameters
`0x89`	`I2CStop`	no parameters
`0x8A`	`I2CSend`	data; returns ACK status
`0x8B`	`I2CReceive`	last flag; returns byte
`0x8D`	`UARTInitialise`	no parameters
`0x8E`	`UARTTransmit`	data byte

0x8F	UARTReceiveCount	returns byte count
0x90	UARTReceive	returns byte
0x91	UARTBaud	rate code
0x92	PWMEnable	channel
0x93	PWMDisable	channel
0x94	PWMSetPrescaler	scaler
0x95	PWMSetDuty8	channel, 8-bit value
0x96	PWMSetDuty10	channel, 10-bit value (LSB, MSB)
0x97	ServoEnable	channel
0x98	ServoDisable	channel
0x99	ServoSetPosition8	channel, 8-bit position
0x9A	ServoSetPosition16	channel, 16-bit position (LSB, MSB)
0x9B	ADCSample8	channel; returns 8-bit value
0x9C	ADCSample10	channel; returns 10-bit value
0x9D	DACEnable	no parameters
0x9E	DACDisable	no parameters
0x9F	DACOutput	5-bit value (0-31)

8. Resources & Repository

8.1 Repository Contents

Folder / File	Contents
Example Programs/Python/	RT_IB.py library + LED, ADC/PWM, and I2C Python examples
Example Programs/C#/	C# example projects (ADCandPWM, LEDFlash)
Example Programs/Flowcode/	Flowcode graphical programming examples
Firmware/	Board firmware files for USB and WiFi variants
Flowcode Component/	Flowcode component for drag-and-drop integration
USB Driver/	USB driver installer for Windows
README.md	Quick-start guide and overview

8.2 Links

GitHub Repository: https://github.com/RowlandTechnology/Interface-Board
License: GNU General Public License v3.0 (GPL-3.0)
Firmware Programmer: Microchip PICkit 3 or PICkit 4 (or compatible)