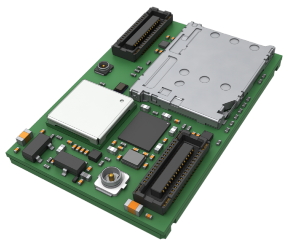


OriginSmart™ Web Specification

Protocol Specification – Software Version
04 04 18 09 33

OriginGPS.com

Table of Contents

LIST OF FIGURES

LIST OF TABLES

TABLE OF REVISIONS

Ver. #	Description	Author/s	Date
1.0	First Edition		May 26, 2020
1.1	Updated Table 2 in Appendix B		June 11, 2020
1.2	Added FDTI driver instructions for Windows and Mac operating systems		March 29, 2021
1.3	Formatted document in new company template		April 29, 2021

ABBREVIATIONS

Abbreviation	Description
APN	Access Point Network
ADC	Analog-to-Digital Converter
APK	Application Kit
DHCP	Dynamic Host Configuration Protocol
ESD	Electronic Sensitive Device
FTDI	Future Technology Devices International
GNSS	Global Navigation Satellite System
GPIO	General Purpose Input/Output
GSM	Global System for Mobile Communications
GUI	Graphical User Interface
I2C	Inter-Integrated Circuit
IMEI	International Mobile Equipment Identity
IMS	IP Multimedia Subsystem
IoT	Internet of Things
LTE	Long-Term Evolution – a standard for wireless broadband communication for mobile devices and data terminals
OriginIoT™	Cellular IoT system to accelerate IoT product development
OriginSmart	Web-based GUI to configure and manage OriginIoT™ systems
RF	Radio Frequency
SMA	Subminiature version A (RF connector)
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver Transmitter
UID	Unique Identifier

RELATED DOCUMENTATION

#	Document Name
1	OriginIoT™ System Datasheet
2	Spider and Hornet - NMEA Protocol Reference Manual
3	OriginIoT™ Development Kit User Guide
4	OriginIoT™ Application Kit User Guide

SCOPE

This document describes the features and specifications of the OriginSmart embedded firmware and the application level communications protocol.

DISCLAIMER

SAFETY INFORMATION

Improper handling and use can cause permanent damage to the product.

ESD SENSITIVITY

This product is an ESD-sensitive device and must be handled with care.

CONTACT INFORMATION

contactus@origingps.com www.origingps.com

1. General

1.1. About the OriginIoT™ System

The OriginIoT™ system is an analytical, customizable system that collects data from sensors. The data can be transferred to a remote server or cloud platform by the OriginIoT™ system via wireless cellular communication (GSM or LTE).

The multi-purpose OriginIoT™ system can accommodate peripheral devices such as sensors or other components via UART, SPI, I²C, or GPIO, and combines cellular communication modules according to customer choice, with superior positional accuracy of stand-alone GNSS. Peripheral devices are configured over a Web interface, eliminating additional embedded firmware effort. The ease and flexibility of utilizing the OriginIoT™ system as a basis for a vast range of applications quickens time to market while minimizing the size of your IoT sensor device.

OriginIoT™ devices enable developers to develop IoT products without writing a single line of embedded code and without RF engineering. A new rapid product cycle is created, dramatically cutting development resources.

2. DESCRIPTION

2.1. System View

The OriginIoT™ system uses the ST-Micro STM32L476JG MCU as the main platform interfacing with the cellular module and provides serial interfaces to control external devices.

The MCU supports multiple interface types on its available physical pinout, as illustrated here:

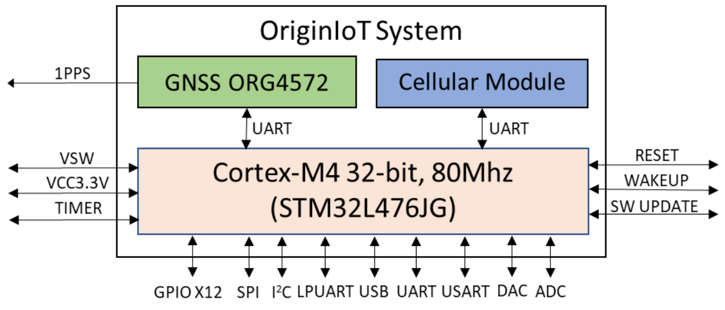


Figure 1: High Level Functional Block Diagram

The OriginSmart firmware resides in the MCU and following cold and warm restarts, the module starts up and immediately connects to the cloud application over the cellular module, and the console terminal connects over the RS-232 port. When the module is connected to the end applications (cloud application or console terminal), then it can be configured by the corresponding end application.

The main functions of the OriginSmart firmware include:

- Communication link with the end application
- Routing the device data to the end application
- Retrieving the data from devices upon request from end applications based on the interface ID
- Delivering the received device data from the end application to the corresponding device based on the interface ID
- Streaming or logging data from devices

Each interface has a unique ID number that enables the end application to communicate and exchange messages with the corresponding interface (GNSS, cellular, MCU and external devices). The embedded software configures the interface types and parameters and assigns a unique ID for each interface for identification purposes, as illustrated in the following OriginIoT™ system interface table:

Table 1: OriginIoT™ System Interface Table

Interface Type	I/O	Interface ID	Attached Device	Connector	PIN
Internal MCU	IO	IF_00	For internal MCU management	Internal	Internal
2wire UART	IO	IF_01	To interface to console terminal	J1	14,16
2,4wire UART	IO	IF_02	Serial interface to cellular module	Internal	Internal
4wire UART	IO	IF_03	Serial interface to GNSS module	Internal	Internal
2wire LPUART	IO	IF_04	Serial interface to external device*	J1	15,17
I2C Master	IO	IF_05	Serial interface to external devices	J1	9,11
GPIO Bitport	O	IF_06	Internal GPIO for cellular reset	Internal	Internal
GPIO Bitport	O	IF_07	Internal GPIO for GNSS reset	Internal	Internal
WAKEUP_1_PIN	I	IF_08	Interrupt interface*	J1	23
WAKEUP_2_PIN	I	IF_09	Interrupt interface*	J1	25
GPIO_IN_1	I	IF_0A	GPIO for external device	J2	6
GPIO_IN_2	I	IF_0B	GPIO for external device	J2	11
GPIO_OUT_1	O	IF_0C	GPIO for external device	J2	7
GPIO_OUT_2	O	IF_0D	GPIO for external device	J2	5
Timer Bitport	O	IF_0F	Timer bitport for external device*	J2	12
DAC Bitport	O	IF_12	Digital to analog convertor for external device*	J2	17
ADC Bitport	I	IF_13	Analog to digital convertor for external device*	J2	23
GPIO Reserved	I/O	IF_14…IF_1F	GPIO for external device*	J2	TBD
2wire UART	I/O	IF_20	Serial interface to external device	J1	18, 22
SPI	I/O	IF_21	Serial interface to external devices*	J1	1, 3, 5
USB 2.0 OTG	I/O	IF_22	Serial interface to external device*	J1	2, 4

*Not supported by OriginSmart firmware.

2.2. Cellular Application Network

The OriginIoT™ system receives a private IP address from the mobile operator and operates as a mobile-originated application. It advertises its private IP address and initiates the connection with the end application.

The application's IP address and server information is stored in the NVM. Refer to the OriginIoT™ Application Kit User Guide for information on how to change these configurations. Ask your OriginGPS representative to set the requested network configurations for you if you are not using an OriginIoT™ Application Kit.

A direct IP socket is established between applications and the OriginIoT™ system described below:

1. Module retrieves the application IP address from its NVM.
2. Module establishes a direct IP socket with the application.
3. Module sends its identification data (MODULE_REGISTER_NOTFY) to the Application. (New message from the module).
4. The module must keep the direct IP sockets functioning, which requires one message (MODULE_APP_KEEPALIVE_NOTFY). (One New Message from the module to the application).
5. The application can send the LoopBack message (MODULE_APP_SET_LB) and the module echoes back (MODULE_APP_SET_LB_RSP). (New set message from the application to the module, new response message from the module to the application, to echoing).

   Upon module power cycling and link down, steps 1 to 5 are repeated.

2.3. Communication Specification

The OriginSmart firmware interfaces with the cloud application through TCP/IP messaging, which is supported by the cellular module. The firmware encapsulates the header message (source and destination address of the interfaces), the device header, and the Device Raw Data to the cellular AT command. The cellular module then sends the data over TCP/IP protocol to the cloud application. Following the received direction from the cloud application, the GSM module transports AT-based messages to the software and based on the destination device interface ID, the firmware routes the device data to the corresponding device.

2.4. Message Format

This section defines and details the OriginSmart message format used to communicate with devices (Cellular, GNSS, external devices, MCU) and the end application.

The message format includes a header and device header and Device Raw Data segments. The header section includes the destination and source interfaces, message length count, and whether an acknowledgement or response is expected. The device header and the raw data segments include data that is device-specific.

2.4.1. Basic Message Format

Figure 2 illustrates the structure of the message interface between the module and the end application.

Count	SRC IF	DES IF	GET	ExpACK	ExpNumByte	Device Header	Device Raw Data

Count	SRC IF	DES IF	SET	ExpACK	ExpRSP	Device Header	Device Raw Data

Count	SRC IF	DES IF	RSP	ExpACK	Device Header	Device Raw Data

Count	SRC IF	DES IF	ACK	Device Header	Device Raw Data

Count	SRC IF	DES IF	NTFY	ExpACK	Device Header	Device Raw Data

Count	SRC IF	DES IF	Multi GET	ExpACK	Device Header	Device Raw Data

Count	SRC IF	DES IF	Multi RSP	ExpACK	Device Header	Device Raw Data

Figure 2: OriginSmart Firmware Basic Message Format

Table 2: Header Message Format Definition

Byte	Field	Value	Description
1,2	Count	0x0008-0xFFFF	Message length
3	SRC IF	0x00-0x7F	The interface ID that initiates the message.
4	DES IF	0x00-0x7F	The destination interface ID where the message is delivered.
5	MSG Type	0x01	**NTFY**: Upon detection of event, the device sends a Notify message to the end application. The data details are in the Device Raw Data segment.
		0x02	**GET**: End application GET message that sends a read request to the corresponding device. The requested reading parameters are retrieved from the Device Raw Data segment.
		0x04	**SET**: End application SET message that sends a write request to the device. The writing data parameters are retrieved from the Device Raw Data segment.

Byte	Field	Value	Description
		0x08	**ACK**: ACK message by the device to acknowledge the received Get and Set command from the end application. ACK message by the end application to acknowledge the received Notify command from the device.
		0x10	**RSP**: RSP message can be initiated by the device to respond to the GET message from the end application. The raw data is included in the Device Raw Data segment
		0x14	**Multi GET***: Multi GET sends a message from the end-application to the module to start/stop streaming data from up to three devices on the I2C bus to the end application
		0x16	**Multi RSP***: Multi RSP is issued by the module in response to the Multi GET command.
6	ExpAck	0x20=NO 0x21=YES	Indicates to the other party if an ACK message is expected.
7	ExpRSP / ExpNumBYTE	0x40=NO 0x41=YES 0x01-0xFF	Indicates to the other party if an RSP message is expected. Only for GET type, the byte 7 is dedicated to set the expected number of bytes that the end application expects from the device.

*Applicable only for I2C bus (IF_05)

The device header segment indicates to the OriginSmart firmware to follow the correct instructions to exchange messages with the devices. The GNSS module works ex-protocol, see description in section 14.

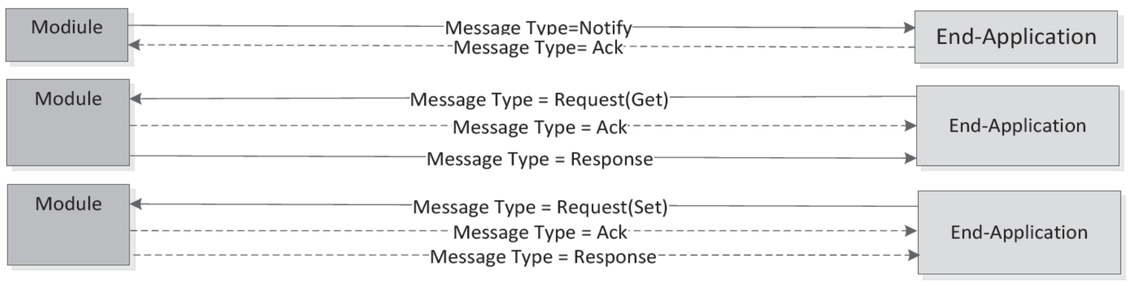


Figure 3: Header Message Flowcharts

2.4.2. UART Based Interface

The UART/USART/LPUART interface type does not have a device header segment and the OriginSmart firmware only processes the Interface ID=0, which covers MCU internal messages. For all other UART/LPUART-based interfaces, the raw data is routed to the corresponding devices without further processing.

Count	SRC IF	DES IF	GET	ExpACK	ExpNumByte	Data0	Data1	...	DataN

Count	SRC IF	DES IF	SET	ExpACK	ExpRSP	Data0	Data1	...	DataN

Count	SRC IF	DES IF	RSP	ExpACK	Data0	Data1	...	DataN

Count	SRC IF	DES IF	ACK	Data0	Data1	...	DataN

Count	SRC IF	DES IF	NTFY	ExpACK	Data0	Data1	...	DataN

Figure 4: Message Format for UART-Based Interface

The following table indicates the commands to control the MCU (DES_IF=0x00).

Table 3: MCU UART-Based Message Definition

Byte	Field	Value	Description
Back from Reset Notification			
0,1	BACKFROMRESET_NTFY	0x0002	Notify the end application when a device comes back from a reset
02	Type	0x00=Warm 0x01=Cold	
Parameter Configuration for GPIO Type Interface			
0,1	IF_PARAM_SET	0x0006	End application request to set the interface parameters
2	IF_ID	0x00-0xFF	Interface ID
3	IF_TYPE	0x01=GPIO	For GPIO Type
4	IF_DIR	0x00=Input 0x01=Output	Interface direction

Byte	Field	Value	Description
5	IF_OP_Mode	0x00=Assigned 0x01=Pass-through	**Assigned:** Used as an additional signal for the serial interface **Pass-through**: The value of the GPIO that is sent to the end-application
Parameter Configuration for UART Type Interface			
0,1	IF_PARAM_SET	0x0006	End application request to set the interface parameters
2	IF_ID	0x00-0xFF	Interface ID
3	IF_TYPE	0x02=UART	For UART Type
4	BaudRate	0x00=1200 0x01=4800 0x02=9600 0x03=19200 0x04=38400 0x05=57600 0x06=115200 0x07=auto-baud Mode 1 0x08=auto-baud Mode 2 0x09=auto-baud Mode 3	Auto baud Mode 1 uses a sync byte of single bit 1. Auto baud Mode 2 uses a sync byte 0x7f. Auto baud Mode 3 uses a sync byte 0x53. Baud rate defaults to 57600 if an Autobaud mode is chosen.
5	FlowCtrl	0x00=Disable 0x01=RTS 0x02=CTS 0x03=RTS and CTS	
6	StopBit	0x00=1 stop bit 0x01=2 stop bits	
7	Parity	0x00=None 0x01=Even 0x02=Odd	
8	Data bits	0x00=7-bit 0x01=8-bit 0x02=9-bit	
9	DataRDY_ID	0x00-0xFF=IF_ID	This indicates the corresponding GPIO interface. For IF_ID=0x00 is indicated, no GPIO is assigned.

Byte	Field	Value	Description
A	DataRDY_ACTION	0x00=Internal 0x01=Pass-through	**Internal**: Upon logic level changes or falling, raising edge on DataRDY_ID, the firmware reads the UART data. **Pass-through**: Upon logic level changes or falling, raising edge on DataRDY_ID level to end application. It is up to the end application to request to read UART data.
B	DataOverRun_ID	0x00-0xFF=IF_ID	This indicates the corresponding GPIO interface. For IF_ID=0x00 is indicated, no GPIO is assigned.
C	DataOverRun_ACTION	0x00=internal 0x01=pass-through	**Internal**: Upon logic level changes or falling, raising edge on DataRDY_ID, the firmware reads UART data. **Pass-through**: Upon logic level changes or falling, raising edge on DataRDY_ID level to end application. The end application determines whether to request to read the UART data.
Parameter Configuration for I2C Type Interface			
0,1	IF_PARAM_SET	0x0006	End application request to set the interface parameters.
2	IF_ID	0x00-0xFF	Interface ID
3	IF_TYPE	0x03=I2C	For I2C Type
4	BitAddrSize	0x00=7-bit 0x01=10-bit	7-bit and 10-bit addressing mode
5	BitRate	0x00=100 kbit/s 0x01=400 kbit/s 0x02=1000 kbit/s	Bit rate, standard, fast, or fast plus
6	RegType	0x01=BYTE 0x02=WORD 0x03=LWORD	Register type
6+size (1 or 2 bytes)	Device Address	Byte or word based on bitAddrSize	
7+ size (1 or 2 bytes)	Data bits	0x00=7-bit 0x01=8-bit 0x02=9-bit	
8+ size (1 or 2 bytes)	DataRDY_ID	0x00-0xFF=IF_ID	This indicates the corresponding GPIO interface. For IF_ID=0x00 is indicated, no GPIO is assigned.

Byte	Field	Value	Description
9+ size (1 or 2 bytes)	DataRDY_ACTION	0x00=Internal 0x01=Pass-through	**Internal**: Upon logic level changes or falling, raising edge on DataRDY_ID, the firmware reads UART data **Pass-through**: Upon logic level changes or falling, raising edge on DataRDY_ID level to end application. The end application determines whether to request to read UART data.
A+ size (1 or 2 bytes)	DataOverRun_ID	0x00-0xFF=IF_ID	This indicates the corresponding GPIO interface. For IF_ID=0x00 is indicated, no GPIO is assigned.
B+ size (1 or 2 bytes)	DataOverRun_ACTION	0x00=internal 0x01=pass-through	**Internal**: Upon logic level changes or falling, raising edge on DataRDY_ID, the firmware reads UART data **Pass-through**: Upon logic level changes or falling, raising edge on DataRDY_ID level to end application. The end application determines whether to read UART data.
Interface Parameter Configuration Response			
0,1	IF_PARAM_SET_RSP	0x0007	Response from MCU to the IF_PARAM_SET based on IF_ID.
2	IF_ID	0x00-0xFF	Interface ID
3	IF_TYPE	0x01=GPIO 0x02=UART 0x03=I2C	
4	State	0x00=Fail 0x01=Completed	Confirms that the IF_PARAM_SET has been successfully executed.
Module Registration Notification			
0,1	MODULE_REGISTER_NOTFY	0x0060	OriginIoT™ system sends a Notify message to register at server.
2…15	IMEI	14 Bytes	IMEI number of the module
16	Value	0x00=Unregister 0x01=Register	
Keep Alive Notification			
0,1	MODULE_APP_KEEPALIVE_NOTFY	0x0063	OriginIoT™ system sends a keepalive message to the application
Loopback Test			
0,1	MODULE_APP_SET_LB	0x0064	Application request to echo back from the module

Byte	Field	Value	Description
2…	Data		
Loopback Test Response			
0,1	MODULE_APP_SET_LB_ RSP	0x0065	Response from the module to MODULE_APP_SET_LB message
2…	Data		

The following table describes the commands to control the devices (GNSS, external devices)

Table 4: Device UART-Based Message Definition

Byte	Field	Value	Description
0	Data0	0x00-0xFF	Device raw data
01…0N	Data1…DataN	0x00-0xFF	Device raw data

2.4.3. I2C-BASED INTERFACE

The I2C-Based Interface type contains the device header and raw date segments. OriginSmart™ firmware exchanges the raw data with the devices based on the device header segment.

Count	SRC IF	DES IF	GET	ExpACK	ExpNumByte	Dev_Add	Reg Start

Count	SRC IF	DES IF	SET	ExpACK	ExpRSP	Dev_Add	Reg Start	NumBytes	Data0	Data1	DataN

Count	SRC IF	DES IF	RSP	ExpACK	Dev_Add	Reg Start	NumBytes	Data0	Data1	DataN

Count	SRC IF	DES IF	ACK	Dev_Add	Reg Start

Count	SRC IF	DES IF	NTFY	ExpACK	Dev_Add	Reg Start	NumBytes	Data0	Data1	DataN

Count	SRC IF	DES IF	Multi GET	ExpACK	Enable	NumD evices	Rate	NumSamples	Dev_Add 0	Reg Start0	NumBytes0	Dev_A ddN	Reg StartN	NumB ytesN

Count	SRC IF	DES IF	Multi RSP	ExpACK	Timer	Data0	Data1	DataN

Figure 5: Message Format for I2C-Based Interface

Table 5: Device I2C-Based Message Definition

Size (Bytes)	Field	Value	Description
Based on bitAddrSize	Dev_Add	0x00-0xFF	Device slave address.
Size based on Reg Type	Reg Start		I2C Register address. The first register address to be read/written.
1	NumBytes	0x01-0xFF	Number of bytes of data to be written.
1	Data0… DataN	0x00-0xFF	Device raw data.
1	Enable	0x00=Disable 0x10-0x12=Enable, choose timer	**Disable**: Stops Multi_Get streaming. If Disable=0x00 is used, the remainder of the message fields are not required. **Enable**: Start Multi_Get streaming. **Timers (0, 1, and 2)**: The system internally assigns three timers with which the user can work. One Multi_Get message can collect data from up to three devices on the I2C bus at the same data rate. If different data rates are required, the user must send multiple Multi_Get messages using different timers. Once a certain timer is chosen, it cannot be used for another message.
1	NumDevices	0x01-0x03	Number of devices in the I2C bus that participate in the Multi_Get message.
2	Rate	0x0001-0x00FF	Data rate in milliseconds. 0x01=one sample per one millisecond. 0xFF=one sample per 256 milliseconds. **Note**: Data rates are highly dependent on the computation load of the system, therefore higher rates are not guaranteed.
2	NumSamples	0x0000=Continuous 0x0001-0xFFFF	Number of samples to be collected and sent.

2.4.4. GPIO-Based Interface

The GPIO interface type does not have any device header segment and the OriginSmart firmware only processes the GPIO-based interface ID and the raw data is routed to the corresponding GPIO ports.

Count	SRC IF	DES IF	GET	ExpACK	ExpNumByte	Data0

Count	SRC IF	DES IF	SET	ExpACK	ExpRSP	Data0

Count	SRC IF	DES IF	RSP	ExpACK	Data0

Count	SRC IF	DES IF	ACK	Data0

Count	SRC IF	DES IF	NTFY	ExpACK	Data0

Byte	Field	Value	Description
0	Data0	(0x00, 0x01)	GPIO Data

Figure 6: Message Format for GPIO-Based Interface

2.5. Non-Volatile Data

2.5.1. Header NVM Data Field

The NVM stores interface parameter settings as shown in

Table 3 (IF_PARAM_SET), otherwise the user does not have access to the NVM. Software and hardware release versions, unique ID, and part numbers are stored in the NVM for internal usage.

2.7.1. Default Configurations

At OriginIoT™ system power up, the firmware applies the default configuration of the MCU interfaces from ID=0x00 to ID=22, as described in Table 1, and enables the corresponding interface drivers' queueing, and message processing.

The default configurations are as follows:

- **ID=0x00 MCU**: No default settings
- **ID=0x01 2 Wire UART Serial Debug Console**: **Baud rate**: 460800, **Data Bits**: 8, **Parity**: None, **Stop Bits**: 1, **Hardware Flow Control**: Off
- **ID=0x02 4 Wire UART Cellular Interface**: **Baud rate**: 230400, **Data Bits**: 8, **Parity**: None, **Stop Bits**: 1, **Hardware Flow Control**: RTS CTS
- **ID=0x03 4 Wire UART GNSS Interface**: **Baud rate**: 19200, **Data Bits**: 8, **Parity**: None, **Stop Bits**:1, **Hardware Flow Control**: None
- **ID=0x04 LPUART 2 Wire Interface**: **Baud rate**: 9600, **Data Bits**: 8, **Parity**: None, **Stop Bits**:1, **Hardware Flow Control**: None
- **ID=0x05 I2C Interface**: 7 bit addressing mode
- **ID=0x06-0x07 Cellular and GNSS reset**: Both set as high-level outputs
- **ID=0x0A-0x0D User Defined GPIO Pins**: 0A and 0B are set as inputs and 0C and 0D are set as outputs.
- **ID=0x20 User Defined UART**: **Baud rate**: 4800, **Data Bits**: 8, **Parity**: None, **Stop Bits**:1, **Hardware Flow Control**: None

2.5.2. Dynamic Configurations

- **MCU**: The end application can set the specific MCU configuration at any time through the messaging format and definitions that are described in Table 3. This dynamic data is stored in the NVM for re-applying configurations with restart cases.
- **GNSS**: The end application is able to configure the GNSS module at any time by sending the GNSS NMEA commands using the messaging format. See Section 14 for details.
- **External Devices**: The end application is able to configure the external devices at any time by sending the corresponding external device commands.

2.6. Restart Strategy

2.6.1. Power-On Reset

When the OriginIoT™ system (MCU) responds following the power-on reset, the OriginIoT™ software applies the default and dynamic MCU configuration data. It then initiates a cold restart on GNSS and cellular modules by writing the dedicated MCU GPIO pins that are connected to GNSS and cellular activation and reset pins. When the connection is re-established with the end application, the OriginIoT™ firmware notifies the end application about the restarted device. The end application then sends the dynamic configuration for the module.

2.6.2. Hardware Reset Pin

The hardware reset pin behaves as described above (Power-on reset).

2.7. GNSS Functionality

Despite being assembled on the module itself, the GNSS module must be regarded as an external device connected to a four-wire UART ID=03. Its operation is then determined by the message format described in this document. Refer to the *Spider and Hornet – NMEA Protocol Reference Manual* document for further information on the relevant NMEA messages for this module.

In regard to the NMEA messages, strip the $ sign and checksum data (checksum enable byte should be set to “enable”), OriginSmart firmware adds this sign and checksum data and sends the complete message to the GNSS module. All NMEA messages must be converted from string to HEX before they are sent to the module.

GNSS is configured to start up and run immediately after module start-up. Its default configuration sends an RMS-type message once per second and transfers it through the cellular module to the end application.

2.8. Sample Messages

2.8.1. External Accelerometer ADXL345

- IF_PARAM_SET for Accelerometer ADXL345 on I2C bus address 3a

00 11 80 00 04 20 41 00 06 05 03 00 00 01 3a 00 00 00 00

00 11	80	00	04	20	41	00 06 05	03 00 00 01 3a 00 00 00 00
count	src	dest	type	ack	rsp	dev h	dev d

This message has to be sent once the data is saved in the NVM and must be restored only after the upload of new firmware.

Response: 00 0B 00 80 10 20 00 07 05 03 01

- ADXL345 set data format, writing the data 01, to register 31

00 0b 80 05 04 20 41 3a 31 01 01

00 0b	80	05	04	20	41	3a	31 01 01
count	src	dest	type	ack	rsp	dev h	dev d

This message data is not stored in the NVM and must be sent again after each time the module is powered up.

Response: 00 09 05 80 10 20 3A 31 01

- Set ADXL to measure mode by sending data 28 to register 2d

00 0b 80 05 04 20 41 3a 2d 01 28

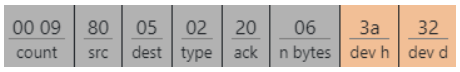
00 0b	80	05	04	20	41	3a	2d 01 28
count	src	dest	type	ack	rsp	dev h	dev d

This message data is not stored in the NVM and must be sent again after each time the module is powered up.

Response: 00 09 05 80 10 20 3A 2D 01

- Read from ADXL one time, read 6 bytes starting at register 32

00 09 80 05 02 20 06 3a 32



00 09	80	05	02	20	06	3a	32
count	src	dest	type	ack	n bytes	dev h	dev d

Response: 00 0F 05 80 10 20 3A 32 06 00 00 00 00 7B 00

The response message includes raw data of the X,Y,Z axes in two byte complements (11 bits). The format can be changed by altering the 31 register byte as shown above. One can see that there are zeroes in the X and Y axis (since the module was stationary on the table when sampled) and Z equals to approximately 1g to represent gravity.

2.8.2. Multi Get

All the devices that participate in the Multi_Get message must be initialized by set messages as shown in the example above. Only I2C devices can participate in Multi_Get messages:

- Start the streaming of accelerometer data at a 256ms frame rate

00 0e 80 05 14 20 10 01 ff 00 00 ff 3a 32 06

Responses:

00 0F 05 80 16 20 00 06 00 F9 FF 00 00 7C 00 0A

The last 6 bytes change as you move the module

- Stop streaming of accelerometer data

00 07 80 05 14 20 00