Contents:

Chapter 1

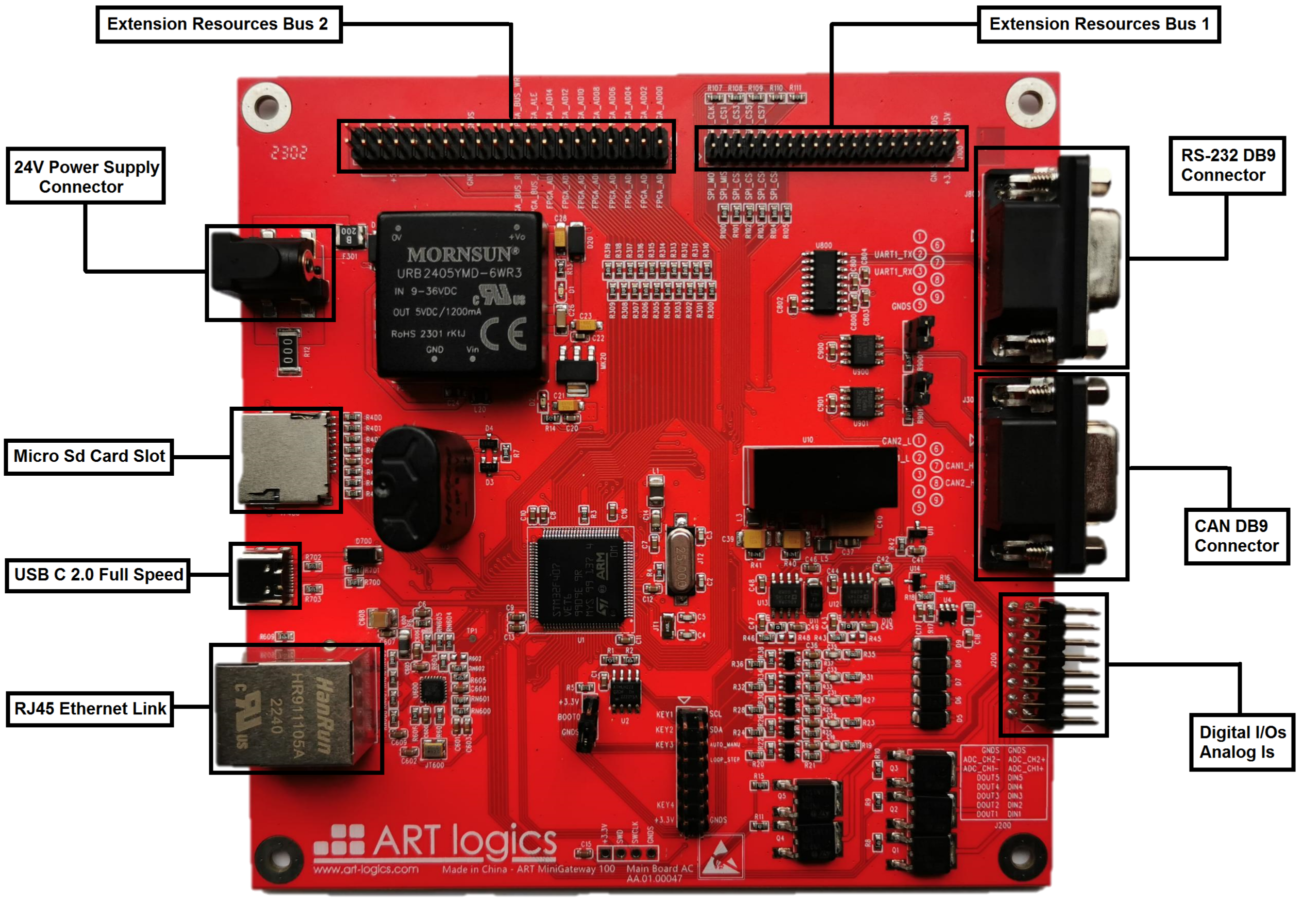
Mini Gateway 100 User Manual

1.1 Introduction

The **Mini Gateway 100** is a Real-Time sequencer and controller, able to run fully configurable test scenarios controlling a wide range of extendable resources, for example CAN buses, Analog, Digital outputs and inputs.

Once the user defined the test sequences, the Mini Gateway 100 Board runs the sequences, by executing actions and sending the results to the host via two interfaces, **USB C** or **Ethernet TCP**. In addition, it is equipped with a **micro Sd Card** to work as a standalone device, without the need of a host.

1.2 Properties



- Real-Time test sequencer with a resolution of 10 ms.
- Supports hardware synchronization.
- USB C 2.0 Full Speed communication interface.
- 100 Mbps Ethernet communication link via TCP.
- 5V voltage supply via USB.
- 24V voltage supply via DC power connector.
- External power supply control via SCPI over RS-232.
- 2 High-speed CAN channels (ISO 11898-2), with configurable Bit rate up to 1 Mbit/s.
- Adjustable CAN termination through a jumper.
- 1 100 Mbps Ethernet link channel with configurable IP addresses, masks and TCP/UDP sockets.
- Built-in support for the SCPI and Modbus protocols.
- 5 Digital input channels.
- 5 Digital output channels.
- 2 Analog differential measurement channels.
- 2 communication buses with extendable resources.

1.3 Getting Started with the Mini Gateway 100

Getting started with the Mini Gateway 100 is easy and straightforward, the user needs to provide a power supply source and a communication link.

Note

In the **standalone mode**, we only need to provide the power supply for the Mini Gateway 100 to work. Check *Standalone Mode* section for more information.

1.3.1 Power Supply

1. Connect a USB C cable from a host to provide the 5V power supply or.
2. Connect a 24V power supply through the DC power connector.
3. Check the LED on the board. If the LED is on, then the board is powered on.

1.3.2 Communication Link

If the USB C cable is used for the communication:

- The device driver provided should be installed.
- Check that the USB COM Port of the device is detected.
- Open the COM Port to start the communication.

If the Ethernet interface is used for the communication:

- Connect the Mini Gateway 100 to your IP local network.
- Optionally, it is useful to ing the IP address to make sure it is connected to the network.
- Establish the TCP connection with the configured IP address and port number.

After establishing the connection with the Mini Gateway 100, send the following command to make sure the communication is working:

```
@11XX_HELLO;
```

The board should say hello back:

```
[23/03/02,09:07:17.0100,0012]#11XX_HELLO;
```

Note

The default IP address of the Mini Gateway 100 is **192.168.0.11** and the port number is **6025**. To Change these parameters, check the *Set the IP Addresses* section.

1.4 Communication Interface

A control host can communicate with the Mini Gateway 100 via the RS-232 connection, the **USB C** 2.0 connection, or over an IP network via **TCP** with the 10/100 Mbps Ethernet link. A defined simple and readable **ASCII protocol** is used for this communication.

1.4.1 RS-232 Communication

The Mini Gateway 100 has one RS-232 port that can be used for the communication with the device. The default parameters of the UART are the following:

Baud
921600

Data Bits
8

Stop Bit
1

Parity
None

1.4.2 USB-C Communication

The Mini Gateway 100 has one USB Port 2.0 type C that can be used for the communication with the device. It is detected as an USB COM Port and can be used with any serial terminal.

Note

Install the provided device driver before you connect the device.

1.4.3 Ethernet TCP Communication

If the Ethernet interface is used for the communication, the default parameters are the following:

Protocol
TCP

IP address
192.168.0.11

IP mask
255.255.255.0

Port number
6025

These parameters are not fixed, and can be changed using commands.

Set the IP Addresses

To change the source, gateway or mask IP address of the device, the user should send the following command.

- **Syntax**:

```
@n_ETH=<IP type>,<IP Address>;
```

- **Description**:
  - <**IP type**>: the type of the IP address to change. A valid IP type is: SOURCE, GATEWAY and MASK.
  - <**IP Adress**>: the new IP address to be set. The format is four decimal values seperated by ..

  The command changes the source IP address of the device to 192.168.0.131.

- **Response**:

```
#n_ETH=<IP type>,<IP Address>;
```

- **Example**:

```
@11_ETH=SOURCE,192.168.0.131;
#11_ETH=SOURCE,192.168.0.131;
```

Get the IP Addresses

To see the current Ethernet configuration of the device with the IP addresses, the user should send the following command.

- **Syntax**:

```
@n_ETH;
```

- **Description**:

  The response of the command contains the type of IP address followed by the current IP address set for the source, gateway and mask addresses.

- **Response**:

```
#n_ETH=SOURCE,<Source IP Address>,GATEWAY,<IP Address>,MASK,<IP Address>;
```

- **Example**:

```
@11_ETH;
#11_ETH=SOURCE,192.168.0.11,GATEWAY,192.168.0.210,MASK,255.255.255.0;
```

1.4.4 Communication Protocol

To send a command to the Mini Gateway 100, the message always starts with '@' and ends with ';', and it includes the ID of the board and the command and its paramaters in the right order.

Once the board receives the command, it checks the ID and if it matches, the board executes the command and sends back a response starting with '#' and ending with ';' with a header including the timestamp and the size of the response.

Command Format:

```
@<ID>_<COMMAND>=<PARAMETERS>;
```

Response Format:

```
[yy/mm/dd,hh:mm:ss.msec,size]#<ID>_<COMMAND>=<RESULT>;
```

The following is an example of a command and response exchange between a host and the Mini Gateway 100, to get the value of the analog input channel number 2.

Command:

```
@1111_GETVOLT=2;
```

Response:

```
[23/08/02,18:27:55.0684,0021]#1111_GETVOLT=2,3.56;
```

1.4.5 Communication Timing

The control host is the master node and regulates the messages rate. To unsure a good use of the board, the user should follow these rules:

- When a command is sent, the next one can be sent only after receiving the reply to the first one.

- If no reply is received after 1.5 seconds, the command is cancelled.

1.4.6 Commands List

The following is the full list of the Mini Gateway 100's commands:

Command	Description
HELLO	Say hello to the board.
SYSID	Get the software version of the device.
PSUV	Set the voltage output value of the power supply connected to the RS-232.
PSUC	Set the current limit value of the power supply connected to the RS-232.
PSU	Turn ON/OFF the power supply connected to the RS-232
PSDV	Get the voltage output value of the power supply connected to the RS-232.
PSUC	Get the current value of the power supply connected to the RS-232.
ETH	Set and get the ethernet's configuration.
RTC	Set and get the real time clock paramters of the board.
STORAGE	A set of commands to manage the Sd Card.
SETDIG	Change the state of the digital output to high.
CLRDIG	Change the state of the digital output to low.
GETDIG	Get the state of the digital input.
CALBRT	Set and get the calibration parameters (Scale, Offset).
GETVOLT	Get the analog measurement channel's voltage value.
SETVOLT	Set the analog output to a voltage value.
OPEN	Open a relay.
CLOSE	Close a relay.
CONFIG	Configure the parameters of the communication channel (CAN, Ethernet).
TSTRT	Validate the configuration.
TSTOP	Reset the configuration.
MSGTX	Send a message in the specified communication channel.
MGSRX	Receive a message in the specified communication channel.

1.5 Power Supply Control

As a compact test solution, the **Mini Gateway 100** offers an RS-232 connector to control an external power supply with support of the **SCPI** interface.

Using the simple defined commands the power supply control, the user is able to change and monitor the voltage and current values during the test.

The RS-232 UART parameters are the following:

Baudrate
  9600

Data Bit
  8

Stop Bit
  1

1.5.1 Set the Output Voltage

This command sets the output voltage of the power supply to the specified value.

- **Syntax**:

```
@11XX_PSUV=<voltage>;
```

- **Acknowledge**:

```
#11XX_PSUV=<voltage>;
```

- **Description**:
  - The <**voltage**> parameter is the power supply's voltage output in volts.
- **Example**:

```
@11XX_PSUV=12.5;
#11XX_PSUV=12.5;
```

The command sets the voltage output to 12.5V.

1.5.2 Set the Output Current Limit

This command sets the output current limit of the power supply to the specified value.

- **Syntax**:

```
@11XX_PSUC=<current>;
```

- **Acknowledge**:

```
#11XX_PSUC=<current>;
```

- **Description**:
  - The <**current**> parameter is the power supply's current limit output in amps.
- **Example**:

```
@11XX_PSUC=1.0;
#11XX_PSUC=1.0;
```

The command sets the current limit output to 1.0A.

1.5.3 Get the Output Voltage

This command gets the output voltage of the power supply.

- **Syntax**:

```
@11XX_PSDV;
```

- **Acknowledge**:

```
#11XX_PSDV=<voltage feedback>;
```

- **Description**:
  - The <**voltage feedback**> is the power supply's voltage output feedback in volts.
- **Example**:

```
@11XX_PSDV;
#11XX_PSDV=23.4;
```

The command gets the voltage output feedback from the power supply 23.4V.

1.5.4 Get the Output Current

This command gets the output current of the power supply.

- **Syntax**:

```
@11XX_PSDC;
```

- **Acknowledge**:

```
#11XX_PSDC=<current feedback>;
```

- **Description**:
  - The <**current feedback**> is the power supply's current output feedback in amps.
- **Example**:

```
@11XX_PSDV;
#11XX_PSDV=5.39;
```

The command gets the current output feedback from the power supply 5.39A.

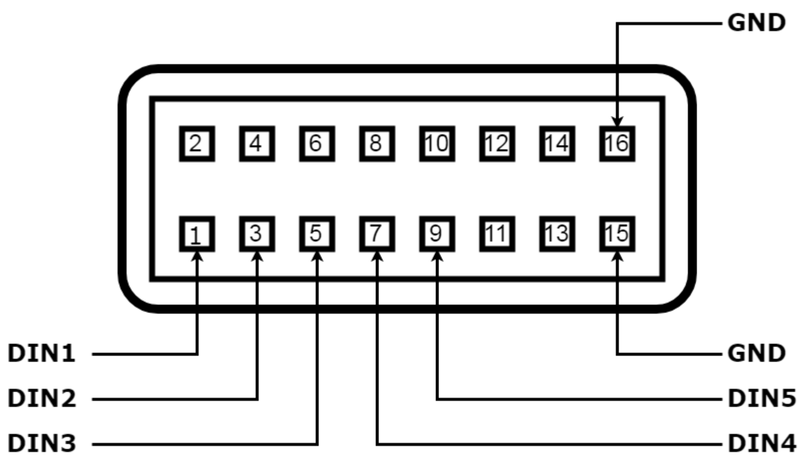
1.6 Resources

The **Mini Gateway 100** offers a set of resources to be used and controlled in the test sequences defined by the user. Also, it is possible to extend the resources if needed through the two **extension buses**. The following table represents the ready to use resources, as well as the extendable ones:

Resource	Number of channels	Extension channels
Digital Input	5	None
Digital Output	5	None
Analog Input	2	48
Analog Output	None	48
Relay	None	96
CAN	2	None
Ethernet	1	None

1.6.1 Digital Input

The Mini Gateway 100 offers **5 Digital Inputs** as resources, accessible through the 16 pins connector **J200** on the front side of the board.



Parameters

- **Input impedance**: 100 kOhms.
- **Input voltage range**: -50 V to 50 V.
- **Threshold**: 2.5 V.
- **Hysteresis**: 17 mV typical.

All the digital inputs share the same ground return, connected to the pins **15** and **16**.

Reading the Digital State

To get the state of the digital input, the user needs to send the following command:

- **Syntax**:

```
@1111_GETDIG=<channel>;
```

- **Acknowledge**:

```
#1111_GETDIG=<channel>,<state>;
```

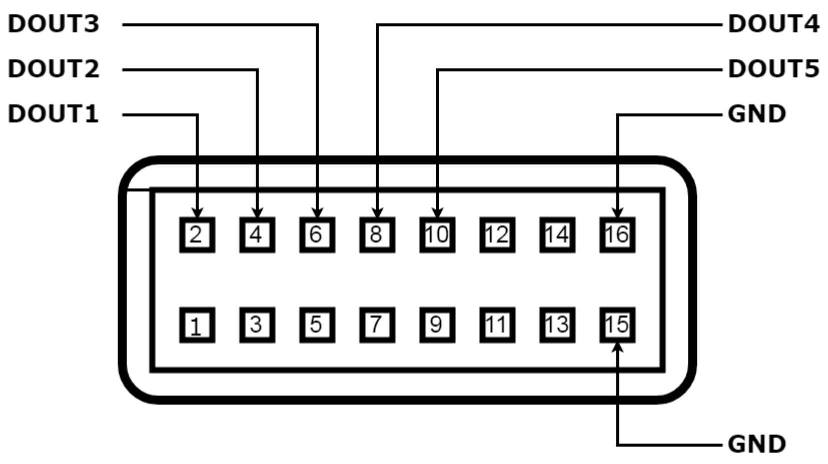
- **Description**:
  - <**channel**>: This parameter defines the digital input channel in the Mini Gateway 100 to access for reading. Valid value is from **1** to **5**.
  - <**state**>: The acknowledge returns the state of the digital input requested. **1** means high state and **0** means low state.
- **Example**:

```
@1111_GETDIG=3;
#1111_GETDIG=3,1;
```

The digital input channel **3** is in **high** state.

1.6.2 Digital Output

The Mini Gateway 100 offers **5 Digital Onputs** as resources, accessible through the 16 pins connector **J200** on the front side of the board.



Parameters

The digital output is an open drain driver, and it is protected for short circuit to positive battery and over temperature.

- **Output impedance**: 60 mOhms.
- **Off leakage current**: 75 µA.
- **Maximum voltage**: 35 VDC.
- **Maximum current**: 2.0 A.

All the digital outputs share the same ground return, connected to the pins **15** and **16**.

Set the Digital Output State

To set the state of the digital output to **high**, the user needs to send the following command:

- **Syntax**:

```
@1111_SETDIG=<channel>;
```

- **Acknowledge**:

```
#1111_SETDIG=<state mask>;
```

- **Description**:
  - <**channel**>: This parameter defines the digital output channel in the Mini Gateway 100 to set to **high**. Valid value is from **1** to **5**.
  - <**state mask**>: The acknowledge returns the state of all digital outputs in hexadecimal format starting with '0X'. The bit 0 is corresponding to the channel 1, and the bit 4 is corresponding to the channel 5; with **1** for a **high** state and **0** for a **low** state.
- **Example**:

```
@1111_SETDIG=2;
#1111_SETDIG=0X13;
```

The value returned *0X13* in hexadecimal format, which is *10011* in binary format indicates:

Channel	State
1	high
2	high
3	low
4	low
5	high

Clear the Digital Output State

To set the state of the digital output to **low**, the user needs to send the following command:

- **Syntax**:

```
@1111_CLRDIG=<channel>;
```

- **Acknowledge**:

```
#1111_CLRDIG=<state mask>;
```

- **Description**:

  - <**channel**>: This parameter defines the digital output channel in the Mini Gateway 100 to set to **low**. Valid value is from **1** to **5**.
  - <**state mask**>: The acknowledge returns the state of all digital outputs in hexadecimal format starting with '0X'. The bit 0 is corresponding to the channel 1, and the bit 4 is corresponding to the channel 5; with **1** for a **high** state and **0** for a **low** state.

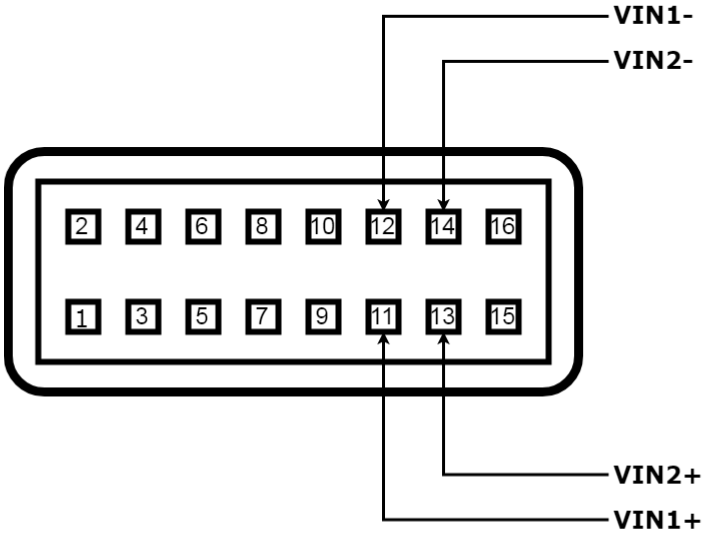
- **Example**:

```
@1111_CLRDIG=5;
#1111_SETDIG=0X0F;
```

The value returned *0X0F* in hexadecimal format, which is *01111* in binary format indicates that the channel **5** is in **low** state, and other channels are **high**.

1.6.3 Analog Input

The Mini Gateway 100 offers **2 Analog Inputs** as differential measurement resources, accessible through the 16 pins connector **J200** on the front side of the board. Also, **48** channels are available by adding the **A20** board.



Parameters

The Analog Input channels available on the board have the following parameters:

- **Input impedance**: 100 kOhms.
- **Maximum differential voltage**: **-5 V to 5 V**.
- **Accuracy**: 5 mV.
- **Resolution**: 12 bits, 1LSB = 0.8 mV.

If extended channels are added, refer to the datasheet of the extension board for the parameters.

Read the Analog Input

To get the voltage value of the analog input, the user needs to send the following command:

- **Syntax**:

```
@1111_GETVOLT=<channel>;
```

- **Acknowledge**:

```
#1111_GETVOLT=<channel>,<voltage>;
```

- **Description**:
  - <**channel**>: This parameter defines the analog input channel to access for measurement. Valid value is from **1** to **2** or from **1** to **50** if the extension board is added.
  - <**voltage**>: The acknowledge returns the voltage value of the analog input requested in volts.
- **Example**:

```
@1111_GETVOLT=2;
#1111_GETVOLT=2,3.502;
```

The voltage value of the analog channel **2** is **3.502 V**.

Calibrate the Analog Input

To reach a maximum accuracy of the analog channels, the user can calibrate it through commands:

- **Syntax**:

```
@1111_CALBRT=VIN,<channel>,<parameter>,<value>;
```

- **Acknowledge**:

```
#11XX_CALBRT=VIN,<channel>,<parameter>,<value>;
```

- **Description**:
  - <**channel**>: This parameter defines the analog input channel to calibrate. Valid value is from **1** to **2** or from **1** to **50** if the extension board is added.
  - <**parameter**>: *FS* for the **scale** calibration, and *OF* for the **offset** calibration.
  - <**value**>: The value of calibration to set.
- **Example**:

```
@1111_CALBRT=VIN,2,FS,1.238;
#1111_CALBRT=VIN,2,FS,1.238;
@1111_CALBRT=VIN,2,OF,-0.6;
#1111_CALBRT=VIN,2,OF,-0.6;
```

The calibration parameters of the analog channel **2** are set to the **scale** value **1.238**, and **offset** value **-0.6**.

1.6.4 Analog Output

The Mini Gateway 100 does not have an Analog Output resource. But, the user can add up to **48** channels by adding the **V10** extension board.

Parameters

The parameters depend on the hardware version of the extension board used. Refer to its datasheet.

Set the Analog Output

To set a voltage value on the Analog Output of the extension board connected to the Mini Gateway 100, the user needs to send the following command:

- **Syntax**:

```
@1111_SETVOLT=<channel>,<voltage>;
```

- **Acknowledge**:

```
#1111_SETVOLT=<channel>,<voltage>;
```

- **Description**:
  - <**channel**>: This parameter defines the analog output channel to set. Valid value is from **1** to **48**.
  - <**voltage**>: The voltage value to set in volts.
- **Example**:

```
@1111_SETVOLT=26,15.78;
#1111_SETVOLT=26,15.78;
```

The voltage value of the Analog Output channel number **26** is set to **15.78 V**.

Calibrate the Analog Output

To reach a maximum accuracy of the analog channels, the user can calibrate it through commands:

- **Syntax**:

```
@1111_CALBRT=VOUT,<channel>,<parameter>,<value>;
```

- **Acknowledge**:

```
#1111_CALBRT=VOUT,<channel>,<parameter>,<value>;
```

- **Description**:
  - <**channel**>: This parameter defines the analog output channel to calibrate. Valid value is from **1** to **48**.
  - <**parameter**>: *FS* for the **scale** calibration, and *OF* for the **offset** calibration.
  - <**value**>: The value of calibration to set.
- **Example**:

```
@1111_CALBRT=VOUT,40,FS,1.0;
#11_CALBRT=VOUT,40,FS,1.0;
@1111_CALBRT=VOUT,40,OF,0.0;
#11XX_CALBRT=VOUT,40,OF,0.0;
```

The calibration parameters of the Analog Output channel **40** are set to the **scale** value **1.0**, and **offset** value **0.0**.

1.6.5 Relay Control

We can add up to **96** relays to the Mini Gateway 100 as resources, by connecting the **RM10** extension board.

Parameters

The parameters depend on the hardware version of the extension board used. Refer to its datasheet.

Open a Relay

Send the following the command to open one or more relays:

- **Syntax**:

```
@1111_OPEN=R<channel1>[,<channel2>,...];
```

- **Acknowledge**:

```
#1111_OPEN=<state mask>;
```

- **Description**:
  - <**channeln**>: This parameter defines the relay's number to open. Valid value is from **1** to **96**.
  - <**state mask**>: The acknowledge returns the state of all relays as a hexadecimal value. **1** represents a closed relay and **0** an open relay.
- **Example**:

```
@1111_OPEN=R15,R65;
#1111_OPEN=0X22E6DF4000000002FD6;
```

The positions **14** and **64** of the mask are set to **0**.

Close a Relay

Send the following the command to close one or more relays:

- **Syntax**:

```
@1111_CLOSE=R<channel1>[,<channel2>,...];
```

- **Acknowledge**:

```
#1111_CLOSE=<state mask>;
```

- **Description**:
  - <**channeln**>: This parameter defines the relay's number to close. Valid value is from **1** to **96**.
  - <**state mask**>: The acknowledge returns the state of all relays as a hexadecimal value. **1** represents a closed relay and **0** an open relay.
- **Example**:

```
@1111_CLOSE=R1,R7;
#1111_CLOSE=0X41;
```

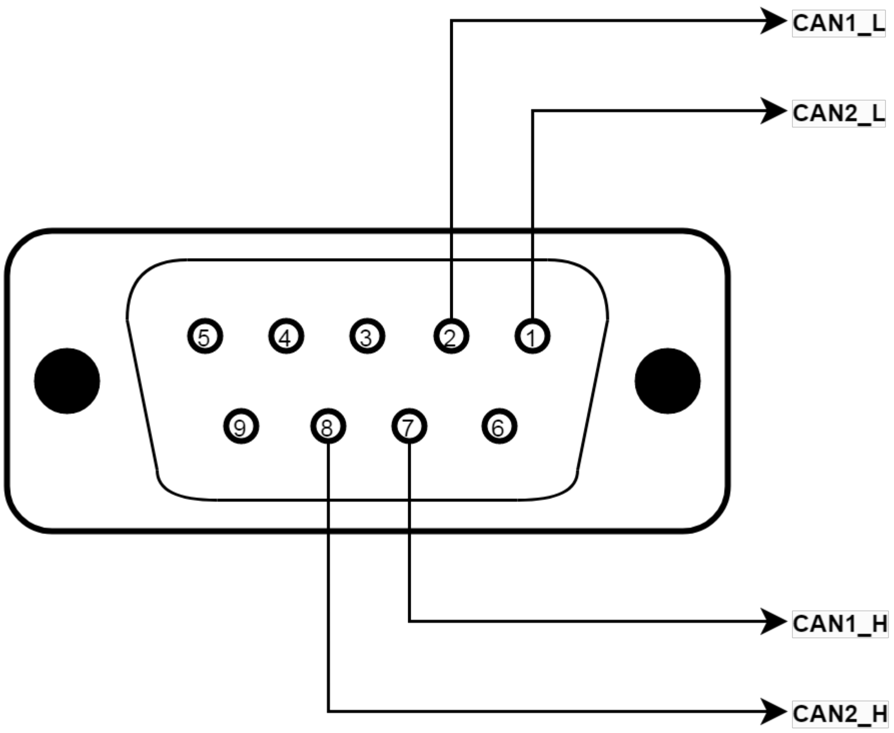
The positions **0** and **6** of the mask are set to **1**.

1.6.6 CANbus

The Mini Gateway 100 has 2 High-Speed CAN connections (ISO 11898-2), compliant with the CAN specifications ISO 11898-1. The pins of the 2 CAN buses are accessible through the DB-9 female connector on the front of the board.

Using the defined commands, the user can configure the baudrate of the bus, the CAN IDs and the messages to send and receive.

The CAN termination can be activated through the jumper **JP901** for the CAN1, and the jumper **JP900** for the CAN2.



Configure the Baudrate

To select the baudrate of the CAN bus, the user needs to send the following command:

- **Syntax**:

```
@1111_CONFIG=CAN<channel>,BAUDRATE,<baudrate>;
```

- **Acknowledge**:

```
#1111_CONFIG=CAN<channel>,BAUDRATE,<baudrate>;
```

- **Description**:
  - <**channel**>: This parameter defines the CAN bus to configure. Valid value is from **1** to **2**.
  - <**baudrate**>: The value of the baudrate to set. The supported baudrate values are: 10K, 20K, 33.3K, 40K, 83.3K, 100K, 125K, 250K, 500K and 1000K (or 1M).

- **Example**:

```
@1111_CONFIG=CAN1,BAUDRATE,500K;
#1111_CONFIG=CAN1,BAUDRATE,500K;
```

  This command sets the baudrate **500 kbit/s** for the **CAN channel 1**.

Configure the CAN IDs

The user can set an alias with a message configuration, and then use that alias to send and receive messages.

- **Syntax**:

```
@1111_CONFIG=CAN<channel>,<direction>,<alias>,<ID type>,<ID>;
```

- **Acknowledge**:

```
#1111_CONFIG=CAN<channel>,<direction>,<alias>,<ID type>,<ID>;
```

- **Description**:
  - <**channel**>: This parameter defines the CAN bus to configure. Valid value is from **1** to **2**.
  - <**direction**>: The direction of the ID. **TX** for transmission and **RX** for reception.
  - <**alias**>: defines the alias of a CAN message. Any name can be use as an alias to identify this ID. However, the alias name cannot include the space character and its length should be less than 12 bytes.
  - <**ID type**>: defines the standard or extended mode of the CAN message ID. Valid values are **STD** (11 bits) or **EXT** (29 bits).
  - <**ID**>: defines the ID of the CAN message. Valid values are any hexadecimal number preceded by *0X* from 0X00 to 0X7FF in STD and from 0X00 to 0X1FFFFFFF for EXT mode.
- **Example**:

```
@1111_CONFIG=CAN2,TX,CH_TX,STD,0X4F;
@1111_CONFIG=CAN2,TX,CH_TX,STD,0X4F;
@1111_CONFIG=CAN2,RX,CH_RX,STD,0X10;
@1111_CONFIG=CAN2,RX,CH_RX,STD,0X10;
```

These commands configure a transmission standard CAN ID **0x4F**, and a reception standard CAN ID **0x10** for the CAN channel **2**.

Send a CAN Message

The following command is used to send a CAN message in the specified CAN channel, with the specified CAN ID:

- **Syntax**:

```
@1111_MSGTX=CAN<channel>,<alias>,<message>;
```

- **Acknowledge**:

```
#1111_MSGTX=CAN<channel>,<alias>,<message>;
```

- **Description**:
  - <**channel**>: This parameter defines the CAN bus used to send the message. Valid value is from **1** to **2**.
  - <**alias**>: must be defined by the CONFIG command before the MSGTX command is used. Its length should be less than 12 bytes.
  - <**message**>: the data bytes to be sent preceded by *0X* (in hexadecimal). The maximum length is 8 bytes.
- **Example**:

```
@1111_MSGTX=CAN1,RqName,0X401100FE;
#1111_MSGTX=CAN1,RqName,0X401100FE;
```

This command sends a CAN message **0x401100FE** with the CAN ID with the alias **RqName** on the CAN channel **1**.

Receive a CAN Message

The reception of the CAN messages has two modes. By default, the device will automatically send the messages received to the host with the following format:

- **Message Format**:

```
#1111_CAN=<channel>,<ID type>,<ID>,<message>;
```

- **Description**:
  - <**channel**>: This parameter defines the CAN bus that received the message. Valid value is from **1** to **2**.
  - <**ID type**>: The standard or extended mode of the CAN message ID. **STD** (11 bits) or **EXT** (29 bits).
  - <**ID**>: The ID of the CAN message received. A hexadecimal number preceded by *0X* from 0X00 to 0X7FF in STD and from 0X00 to 0X1FFFFFFF for EXT mode.
  - <**message**>: the data bytes received preceded by *0X* (in hexadecimal). The maximum length is 8 bytes.
- **Example**:

```
#1111_CAN=1,STD,0XF0,0X3FEE45;
```

This command sends back the CAN message received with the standard CAN ID with 0xF0 on the CAN channel **1**.

In case the user defined an alias for the ID, the message is saved in the device's memory and then sent back to host upon request using MSGRX command.

The following command is used to get the CAN message received in the specified CAN channel, with the specified CAN ID:

- **Syntax**:

```
@1111_MSGRX=CAN<channel>,<alias>,<size>;
```

- **Acknowledge**:

```
#1111_MSGRX=CAN<channel>,<alias>,<message>;
```

- **Description**:
  - <**channel**>: This parameter defines the CAN bus that received the message. Valid value is from **1** to **2**.
  - <**alias**>: must be defined by the CONFIG command before the MSGTX command is used. Its length should be less than 12 bytes.
  - <**size**>: defines how many bytes needs to read, the maximum number is 8.
  - <**message**>: the data bytes to received preceded by *0X* (in hexadecimal). The maximum length is 8 bytes.
- **Example**:

```
@1111_MSGRX=CAN1,CH13,8;
@1111_MSGRX=CAN1,CH13,0X65012507002098FE;
```

This command sends back the CAN message received with the CAN ID with the alias **CH13** on the CAN channel **1**.

Warning

To use the CAN channel, the user needs to configure CAN parameters and also validate the configuration by sending the **TSTRT** command.

Quick Validation Test

To validate the opration of the CAN channels of the Mini Gateway 100, the user can connect the two channels together and perform a quick send and receive test.

- Connect the CAN1_L **Pin 2** with the CAN2_L **Pin 1**.
- Connect the CAN1_H **Pin 7** with the CAN2_H **Pin 8**.
- Make sure that the CAN termination with the jumper is the same for both channels.
- Configure the same baudrate for CAN1 and CAN2:

```
@1111_CONFIG=CAN1,BAUDRATE,500K;
@1111_CONFIG=CAN2,BAUDRATE,500K;
```

- Configure the transmission and reception IDs:

```
@1111_CONFIG=CAN1,TX,CH1TX,STD,0X11;
@1111_CONFIG=CAN2,RX,CH2RX,STD,0X11;
@1111_CONFIG=CAN2,TX,CH2TX,STD,0XFF;
@1111_CONFIG=CAN1,RX,CH1RX,STD,0XFF;
```

- Validate the configuration:

```
@1111_TSTRT;
```

- Send a message from CAN1 to CAN2, and from CAN2 to CAN1:

```
@1111_MSGTX=CAN1,CH1TX,0X0102030405060708;
@1111_MSGTX=CAN2,CH2TX,0X1122334455667788;
```

- Check the reception of the same messages from CAN1 and CAN2:

```
@1111_MSGRX=CAN1,CH1RX,8;
@1111_MSGRX=CAN2,CH2RX,8;
```

1.6.7 Ethernet Link

Besides using the Ethernet Link as a communication medium between a host controller and the Mini Gateway 100, it can also be used as a resource to control external instruments and devices via **TCP**.

Configure the MAC Address

- **Syntax**:

```
@1111_CONFIG=ETH<channel>,MACADDR,<MAC address>;
```

- **Acknowledge**:

```
#1111_CONFIG=ETH<channel>,MACADDR,<MAC address>;
```

- **Description**:
  - <**channel**>: Ethernet channel to configure. Valid value is 1.
  - <**MAC address**>: defines the MAC Address of the Ethernet interface. MAC Address is 6 bytes in hexadecimal format separated by ':' or '-' characters.
- **Example**:

```
@1111_CONFIG=ETH1,MACADDR,1B:63:0A:8E:00:CE;
#1111_CONFIG=ETH1,MACADDR,1B:63:0A:8E:00:CE;
```

This command sets the Ethernet interface **1** MAC address to 881B:63:0A:8E:00:CE**.

Configure the IP parameters

- **Syntax**:

```
@1111_CONFIG=ETH<channel>,IP4ADDR,<source address>,<mask address>,
<gateway address>;
```

- **Acknowledge**:

```
#1111_CONFIG=ETH<channel>,IP4ADDR,<source address>,<mask address>,
<gateway address>;
```

- **Description**:
  - <**channel**>: Ethernet channel to configure. Valid value is 1.
  - <**source address**>: sets the IP v4 source address of the Ethernet interface. A valid address is 4 bytes in decimal format separated by '.'.
  - <**mask address**>: sets the subnet mask IP v4 address of the Ethernet interface. A valid address is 4 bytes in decimal format separated by '.'.
  - <**gateway address**>: sets the default gateway IP v4 address of the local network. A valid address is 4 bytes in decimal format separated by '.'.
- **Example**:

```
@1111_CONFIG=ETH1,IP4ADDR,192.168.0.11,255.255.255.0,192.168.0.1;
#1111_CONFIG=ETH1,IP4ADDR,192.168.0.11,255.255.255.0,192.168.0.1;
```

This command sets the Ethernet interface **1** IP address to **192.168.0.11**, Mask address to **255.255.255.0** and Gateway address to **192.168.0.1**.

Configure a Client Socket

- **Syntax**:

```
@1111_CONFIG=ETH<channel>,TCP,<socket name>,BIND,<source address>,<source port>,
CONNECT,<destination address>,<destination port>;
```

- **Acknowledge**:

```
#1111_CONFIG=ETH<channel>,TCP,<socket name>,BIND,<source address>,<source port>,
CONNECT,<destination address>,<destination port>;
```

- **Description**:
  - <**channel**>: Ethernet channel to configure. Valid value is 1.
  - <**socket name**>: defines the alias of a socket. Any name can be used as an alias. However, it cannot include the space character and it must be unified, and its length should be less than 12 bytes.
  - <**source address**>: sets the IP v4 source address of the Ethernet interface to be used for this socket.
  - <**source port**>: sets the source port number for this socket.
  - <*destination address*>: sets the IP v4 destination address of the Ethernet interface to be used for this socket.
  - <*destination port*>: sets the destination port number for this socket.
- **Example**:

```
@1111_CONFIG=ETH1,TCP,AGILENT,BIND,192.168.0.15,41569,CONNECT,192.168.0.19,502;
#1111_CONFIG=ETH1,TCP,AGILENT,BIND,192.168.0.15,41569,CONNECT,192.168.0.19,502;
```

This command sets up the parameters of a socket with the source address

Send a Packet

- **Syntax**:

```
@1111_MSGTX=ETH<channel>,<socket name>,<message>;
```

- **Acknowledge**:

```
#1111_MSGTX=ETH<channel>,<socket name>,<message>;
```

- **Description**:
  - <**channel**>: Ethernet channel to send the packet. Valid value is 1.
  - <**socket name**>: the alias of the socket previously defined. The length should be less than 12 bytes.
  - <**message**>: the data bytes to send preceded by *0X* (in hexadecimal). The maximum length is 255 bytes.
- **Example**:

```
@1111_MSGTX=ETH1,GBF,0X2A49444E3F0D0A;
#1111_MSGTX=ETH1,GBF,0X2A49444E3F0D0A;
```

This command sends **IDN?rn** to the socket defined by the alias **GBF**.

Receive a Packet

- **Syntax**:

```
@1111_MSGRX=ETH<channel>,<socket name>,<size>;
```

- **Acknowledge**:

```
#1111_MSGRX=ETH<channel>,<socket name>,<size>;
```

- **Description**:
  - <**channel**>: Ethernet channel that received the packet. Valid value is 1.
  - <**socket name**>: the alias of the socket previously defined. The length should be less than 12 bytes.
  - <**size**>: the maximum data bytes to receive preceded by *0X* (in hexadecimal). The maximum length is 255 bytes.
- **Example**:

```
@1111_MSGRX=ETH1,OSC,255;
#1111_MSGRX=ETH1,OSC,0X312E343537380A0D;
```

This command sends back the data received **1.4578rn** in the socket defined by the alias **OSC**.

1.7 Real Time Clock

The **Mini Gateway 100** integrates a real time clock that can keep the date and time values even after powering off the device. This clock is used mainly to set the timestamp of every device's action and include it in the header of every response. For example, the timestamp of a CAN message reception or a voltage measurement.

1.7.1 Set RTC parameters

To set new parameters of the RTC, the user needs to send this command:

- **Syntax**:

```
@11XX_RTC=SET,<Year>,<Month>,<Day>,<WeekDay>,<Hours>,<Minutes>,<Seconds>;
```

- **Acknowledge**:

```
#11XX_RTC=<Year>,<Month>,<Day>,<WeekDay>,<Hours>,<Minutes>,<Seconds>;
```

- **Description**:
  - <**Year**>: Value less than 99.
  - <**Month**>: Value between 1 and 12.
  - <**Day**>: Value between 1 and 31.
  - <**WeekDay**>: Value between 1 and 7.
  - <**Hours**>: Value between 0 and 23.
  - <**Minutes**>: Value between 0 and 59.
  - <**Seconds**>: Value between 0 and 59.
- **Example**:

```
@1111_RTC=SET,23,8,30,3,8,21,1;
#1111_RTC=23,8,30,3,8,21,1;
```

  After sending this command, the board RTC is set to 23/08/30 08:21:01.

1.7.2 Get RTC parameters

The user can get the parameters of the RTC currently used by the device.

- **Syntax**:

```
@11XX_RTC=GET;
```

- **Acknowledge**:

```
#11XX_RTC=<Year>,<Month>,<Day>,<WeekDay>,<Hours>,<Minutes>,<Seconds>;
```

- **Description**:

  The parameters are the same as the Set RTC command.
- **Example**:

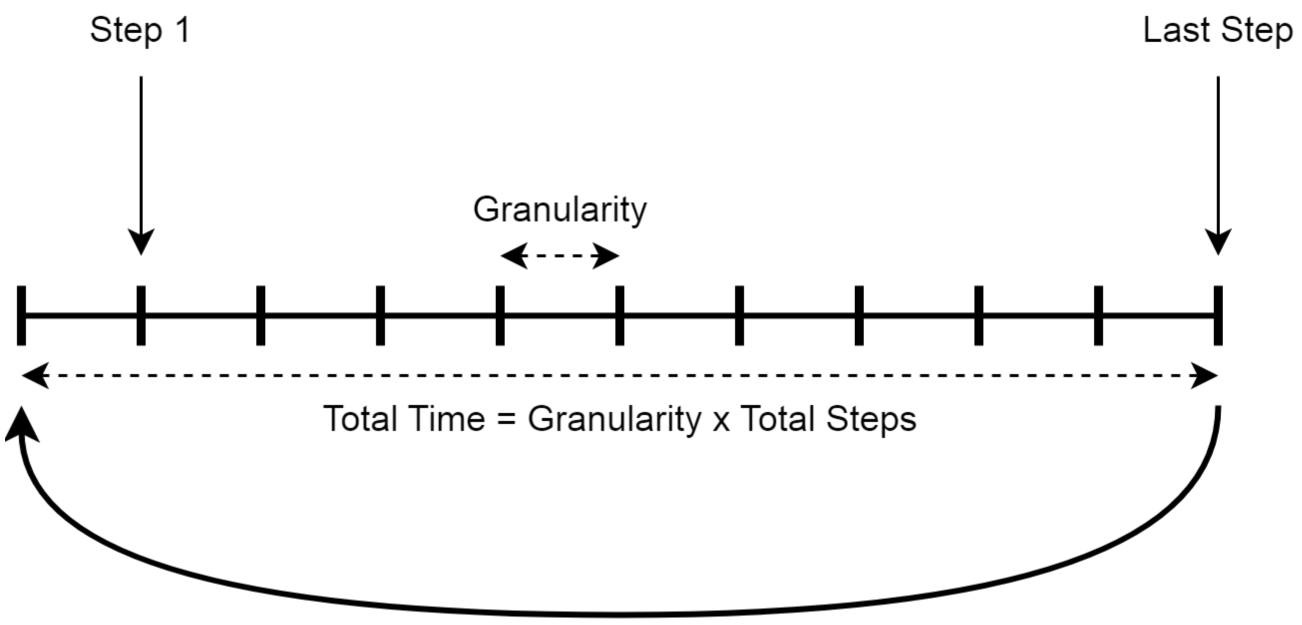
```
@1111_RTC=GET;
#1111_RTC=23,3,2,6,21,24,19;
```

  The response of the board indicates the RTC parameters: 23/03/02 21:24:19.

1.8 Real-Time Testing

The system defined of single commands to control the resources of the Mini Gateway 100 offers an easy and integrated solution to perform automated test sequences. However, this method cannot cope with some applications that require a great timing precision - for example, if an ECU (Electronic Control Unit) interaction has to be captured and responded to in a range of milliseconds. For this purpose, the Mini Gateway 100 includes the **PROCESS** feature.

The **PROCESS** feature allows the user to define the tests to be performed as a sequence of actions with their respective time, and upload them to the board. This way, the system is able to run in a Real-Time environment.



1.8.1 Define a Process

The Mini Gateway 100 can handle up to 32 processes, that can run simultaneously. The process runs in a loop by executing the actions defined in each step, and returning to the first step at the end. The total time that the process loop will last is calculated by multiplying the total number of steps by the granularity.

The results of each loop will be sent automatically to the host controller at the end of the loop, or will be saved on the Sd Card in the standalone mode.

- **Syntax**:

```
@1111_PROCESS=<id>,DEFINE,<granularity>,<totalsteps>;
```

- **Acknowledge**:

```
#1111_PROCESS=<id>,DEFINE,<granularity>,<totalsteps>;
```

- **Description**:
  - <**id**>: defines the process identifier. It is a decimal number, from 1 to 255. 32 processes can be defined in one board at a time.
  - <**granularity**>: defines the minimum time (in ms) resolution of the process, i.e. the time between two steps. It's an integer number. Valid value is 10, 20, 30,. . . ,65530.
  - <**totalsteps**>: defines the total mumber of the steps that the process will endure. Valid value for it is 1 to 4294967295.

- **Example**:

```
@1111_PROCESS=1,DEFINE,10,450;
#1111_PROCESS=1,DEFINE,10,450;
```

This command defines a new process with the identifier **1**, with **450** steps and **10 ms** timing between each step. The process loop will run in total for 4500 ms.

1.8.2 Add a Process Action

After defining the global parameters of a process, the user can actions inside the process. The actions can be any command to control the Mini Gateway 100's resources.

- **Syntax**:

```
@1111_PROCESS=<id>,<step>,<command>;
```

- **Acknowledge**:

```
#1111_PROCESS=<id>,<step>,<command>;
```

- **Description**:
  - <**id**>: defines the process identifier. It is a decimal number, from 1 to 255. The process needs to be defined already.
  - <**step**>: defines the step number when the command will be executed. It is a decimal number that should be equal or less that the total number of steps.
  - <**command**>: defines the operation that board will be executed at the <step>, with its related parameters. the command can be the following commands: CLOSE, OPEN, SETDIG, CLRDIG, GETDIG, SETVOLT, GETVOLT, MSGTX, MSGRX.
- **Example**:

```
@1111_PROCESS=5,23,GETVOLT,2;
#1111_PROCESS=5,23,GETVOLT,2;
```

This command adds an action in the process **5** to execute a **GETVOLT** command at the step number **23**.

1.8.3 Control a Process

Once a process is defined and configured, the user can start, stop and monitor the executing of the process.

- **Syntax**:

  @1111_PROCESS=<id>,END|DELETE|START|STOP|DEFINE;

- **Acknowledge**:

```
#1111_PROCESS=<id>,END|START|STOP;
#1111_PROCESS=<id>,DEFINE,<granularity>,<totalsteps>,LOOP=<loop>;
```

- **Description**:
  - <**id**>: defines the process identifier. It is a decimal number, from 1 to 255. The process needs to be defined already.
  - **END**: informs the board that the definition of the process is done. This command is needed to be sent before starting the process.
  - **DELETE**: used to delete one predefined process or all process. After it is executed successfully, the process will not be available any more.
  - **START**: starts the execution of a predefined process.
  - **STOP**: stops the execution of a process previously started.
  - **DEFINE**: sends back the global parameters of the process with the current loop number.
- **Example**:

```
@1111_PROCESS=2,DEFINE;
#1111_PROCESS=2,DEFINE,10,1000,LOOP=43;
```

The command sends back the parameters of the process **5**: granularity of **10 ms** and a total steps number of **1000** steps. Also, the board executed **42** loops until now, and it is now executing the loop **43**.

1.9 Synchronization

The **Mini Gateway 100** implements a hardware synchronization mechanism, to be used in case the board needs to be synchronized with other systems, or when using more than one board.

To ensure that the test execution is synchronized, the user needs to activate the synchronization mode before starting the operation. The **synchronization mechanism** includes two modes of operation, that can be activated at the same if needed.

Slave Mode: The external system needs to provide a clock signal, of a frequency between 1 kHz and 16 kHz, on the **Digital Input 1** (J200 - Pin 1) of the Mini Gateway 100.

Master Mode: The Mini Gateway 100 outputs a clock signal, of 1 kHz, on the **Digital Output 1** (J200 - Pin 2).

Note

If a Digital Input or Output is used for the synchronization, it is not possible to use it as a resource.

1.9.1 Control the Synchronization

To enable or disable the synchronization signal in the master or slave operation, the user needs to send this command:

- **Syntax**:

```
@1111_SYNCHRO=<mode>,<state>;
```

- **Acknowledge**:

```
#1111_SYNCHRO=<mode>,<state>;
```

- **Description**:
  - <**mode**>: defines the synchronization mode. **OUT** for the master mode, and **IN** for the slave mode.
  - <**state**>: defines the state of the mode. **START** to enable and **STOP** to disable.
- **Example**:

```
@1111_SYNCHRO=OUT,START;
#1111_SYNCHRO=OUT,START;
```

After sending this command, the board outputs a 1 kHz clock signal on the **DOUT1**.

1.9.2 Check the Synchronization Status

The user can send the following command to check the status of the synchronization:

- **Syntax**:

```
@1111_SYNCHRO=<mode>;
```

- **Acknowledge**:

```
#1111_SYNCHRO=<mode>,<state>;
```

- **Description**:

  - <**mode**>: defines the synchronization mode. **OUT** for the master mode, and **IN** for the slave mode.
  - <**state**>: the board sends back the current state of the synchronization mode. **ACTIVE** indicates the synchronization is activated, otherwise it is **INACTIVE**.

- **Example**:

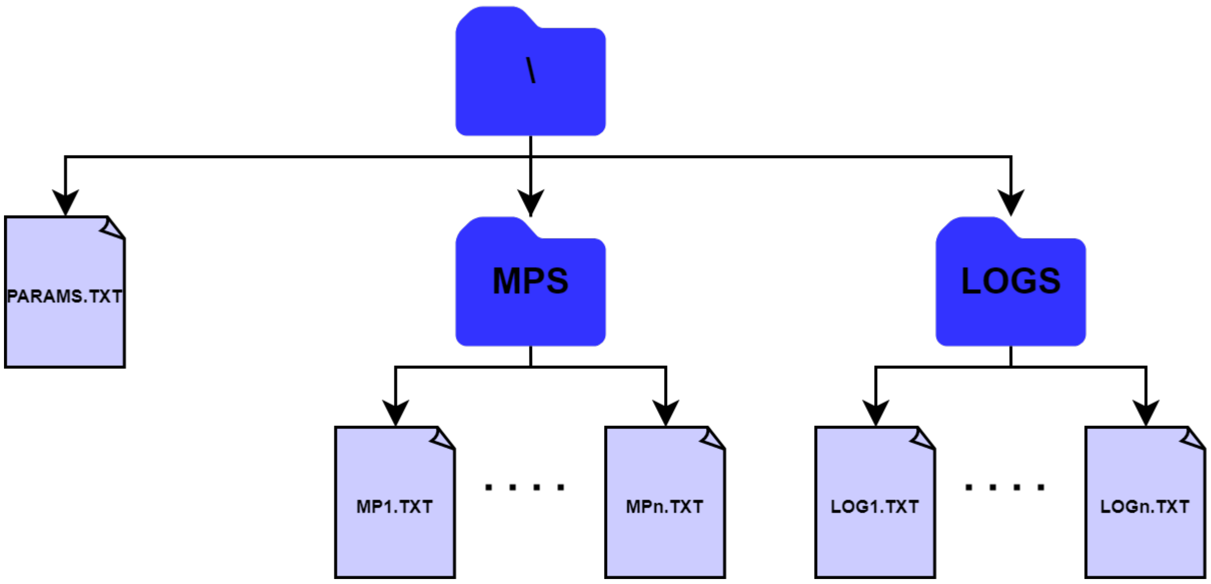
```
@1111_SYNCHRO=IN;
#1111_SYNCHRO=IN,ACTIVE;
```

The response of the board indicates that the synchronization slave mode is active.

1.10 Standalone Mode

the Mini Gateway 100 is able to run in an independant mode without the intervention of a host controller connected to the board. In this Standalone Mode, the Mini Gateway 100 executes the processes defined in the micro Sd Card and saves the results in the form of logs.

The Sd Card file system is organized as a file **PARAMS.TXT** for the global parameters, a folder **MP** that contains the configuration of the processes and a folder **LOGS** that contains the saved results of the processes.



1.10.1 Standalone Parameters

The parameters of the Standalone Mode using the Sd Card are stored in the **PARAMS.TXT** file. These are the configurable parameters:

- **LOGGING**: Enable (=1) or disable (=0) the logging of the results in the **LOGS** folder.
- **AUTO_READ**: Enable (=1) or disable (=0) the automatic loading the configurations of the processes at the startup of the Mini Gateway 100.
- **AUTO_PUSH**: Enable (=1) or disable (=0) the board to push the results on the communication interface.

1.10.2 Test Configuration

The configuration of the processes are saved inside the **MP** folder. The user can manually connect the Sd Card on the computer and save the configuration, or instruct the Mini Gateway 100 to save the commands sent by USB or Ethernet:

```
@1111_STORAGE=DLSTART;
<Processes Config>
@1111_STORAGE=DLSTOP;
```

Inside the folder **MP**, we can find twp types of files:

- **CONFIG.TXT**: contains the commands CONFIG shared by all processes.
- **MP<n>.TXT**: contains the configuration of the process with the identifier n.

1.10.3 Saving the Results

The Mini Gateway 100 saves the results of all the processes running inside the folder **LOGS**, if the parameter **LOGGING** is enabled in the **PARAMS.TXT** file.

Each result is saved in one line following the same format: **<Timestamp>,<Result>\r\n**

1.11 SCPI Command Reference

This section describes the SCPI (Standard Commands for Programmable Instruments) commands supported by the Mini Gateway 100, alongside their legacy equivalents.

1.11.1 General SCPI Rules

When using SCPI commands on the Mini Gateway 100, the following conventions must be respected:

- **Queries use "?"** Only commands ending with a question mark return a response from the device. Example: `DIG:GET? 1` returns the state of digital channel **1**, while `DIG:SET 1` sets digital channel **1** but does not return a value.
- **Command termination** Each SCPI command **must be terminated with a space and a line feed** (`\n`). Example: `DIG:GET? 1 \n`
- **Multiple commands** Several SCPI commands can be concatenated in a single line, separated by semicolons (`;`). Each subcommand still follows the same rules. Example: `CONF:CAN:BAUD 1,500K;DIG:GET? 2\n`
- **Case-insensitive** SCPI commands are not case-sensitive. Both `DIG:GET?` and `dig:get?` are accepted.
- **Optional and short forms** Keywords in SCPI often allow abbreviated forms (uppercase letters indicate the minimum required). For example, `VOLTage` can be written as `VOLT`.
- **Legacy compatibility** For each SCPI command, a legacy format (`@11_...`) is also supported. These are documented alongside the SCPI syntax.

Note

Unless specified otherwise, parameters such as channel numbers start at **1** and are limited to the number of available hardware resources on the Mini Gateway 100.

1.11.2 Identification

*IDN?

- **Syntax**:

```
*IDN?
```

- **Description**:

  Returns the device identification string containing hardware type, hardware version, part number, serial number, and software version.

- **Legacy Equivalent**:

```
@11_SYSID;
```

- **Example**:

```
*IDN?
2000,1.0,120000,23Y0000,MNW100_01_42
```

1.11.3 System Date and Header

SYST:DATE?

- **Syntax**:

```
SYST:DATE?
```

- **Description**:

  Returns the current system date and time in the format: `<year>,<month>,<day>,<weekday>,<hours>,<minutes>,<seconds>`.

- **Legacy Equivalent**:

```
@11_RTC=GET;
```

- **Example**:

```
SYST:DATE?
25,8,26,2,15,30,0
```

SYST:DATE

- **Syntax**:

```
SYST:DATE <year>,<month>,<day>,<weekday>,<hours>,<minutes>,<seconds>
```

- **Description**:

  Sets the system date and time.

- **Legacy Equivalent**:

```
@11_RTC=SET,<year><month><day><weekday><hours><minutes><seconds>;
```

- **Example**:

```
SYST:DATE 25,8,26,2,15,30,0
```

SYST:HEAD

- **Syntax**:

```
SYST:HEAD <ON|OFF>
```

- **Description**:

  Enables or disables the header prefix (date and time) in SCPI responses.

- **Legacy Equivalent**:

```
NONE
```

- **Example**:

```
SYST:HEAD ON
```

1.11.4 Digital IO

DIG:GET?

- **Syntax**:

```
DIG:GET? <channel>
```

- **Description**:

  Reads the state of the specified digital input channel. Returns **0** for low state and **1** for high state.

- **Legacy Equivalent**:

```
@11_GETDIG=<channel>;
```

- **Example**:

```
DIG:GET? 3
1
```

DIG:SET

- **Syntax**:

```
DIG:SET <channel>
```

- **Description**:

  Sets the specified digital output channel to high. Returns the hexadecimal state mask of all digital outputs.

- **Legacy Equivalent**:

```
@11_SETDIG=<channel>;
```

- **Example**:

```
DIG:SET 2
0X13
```

1.11.5 Analog IO

MEAS:CHAN:VOLT?

- **Syntax**:

```
MEAS:CHAN:VOLT? <channel>
```

- **Description**:

  Measures the voltage on the specified analog input channel.

- **Legacy Equivalent**:

```
@11_GETVOLT=<channel>;
```

- **Example**:

```
MEAS:CHAN:VOLT? 2
3.502
```

1.11.6 CANbus

CONF:CAN:BAUD

- **Syntax**:

```
CONF:CAN:BAUD <channel>,<baudrate>
```

- **Description**:

  Sets the baudrate of the specified CAN channel.

- **Legacy Equivalent**:

```
@11_CONFIG=CAN<channel>,BAUDRATE,<baudrate>;
```

- **Example**:

```
CONF:CAN:BAUD 1,500K
```

CONF:CAN:RX

- **Syntax**:

```
CONF:CAN:RX <channel>,<alias>,<STD|EXT>,<ID>
```

- **Description**:

  Configures a CAN reception ID with alias, ID type, and ID value.

- **Legacy Equivalent**:

```
@11_CONFIG=CAN<channel>,RX,<alias>,<ID type>,<ID>;
```

- **Example**:

```
CONF:CAN:RX 1,CH1RX,STD,0XFF
```

CONF:CAN:TX

- **Syntax**:

```
CONF:CAN:TX <channel>,<alias>,<STD|EXT>,<ID>
```

- **Description**:

  Configures a CAN transmission ID with alias, ID type, and ID value.

- **Legacy Equivalent**:

```
@11_CONFIG=CAN<channel>,TX,<alias>,<ID type>,<ID>;
```

- **Example**:

```
CONF:CAN:TX 1,CH1TX,STD,0X11
```

MSGTX:CAN

- **Syntax**:

```
MSGTX:CAN <channel>,<alias>,<message>
```

- **Description**:

  Sends a CAN message through the specified channel and alias.

- **Legacy Equivalent**:

```
@11_MSGTX=CAN<channel>,<alias>,<message>;
```

- **Example**:

```
MSGTX:CAN 1,CH1TX,0X0102030405060708
```

MSGRX:CAN?

- **Syntax**:

```
MSGRX:CAN? <channel>,<alias>,<size>
```

- **Description**:

  Receives a CAN message from the specified channel and alias, up to the defined size.

- **Legacy Equivalent**:

```
@11_MSGRX=CAN<channel>,<alias>,<size>;
```

- **Example**:

```
MSGRX:CAN? 1,CH1RX,8
```

TSTRT

- **Syntax**:

```
TSTRT
```

- **Description**:

  Validates the current CAN configuration and starts CAN communication.

- **Legacy Equivalent**:

```
@11_TSTRT;
```

- **Example**:

```
TSTRT
```

TSTOP

- **Syntax**:

```
TSTOP
```

- **Description**:

  Stops the CAN communication and resets the configuration.

- **Legacy Equivalent**:

```
@11_TSTOP;
```

- **Example**:

```
TSTOP
```

1.11.7 Process

PROC:DEF

- **Syntax**:

```
PROC:DEF <id>,<granularity>,<totalsteps>
```

- **Description**:

  Starts a new process definition for the given process ID, with the specified granularity and total number of steps.

- **Legacy Equivalent**:

```
@11_PROCESS=<id>,DEFINE,<granularity>,<totalsteps>;
```

- **Example**:

```
PROC:DEF 1,10,20
```

PROC:DEF?

- **Syntax**:

```
PROC:DEF? <id>
```

- **Description**:

  Queries the definition of the process ID.

- **Legacy Equivalent**:

```
@11_PROCESS=<id>,DEFINE;
```

- **Example**:

```
PROC:DEF? 1
LOOP=32
```

PROC:END

- **Syntax**:

```
PROC:END <id>
```

- **Description**:

  Ends the definition of the process ID.

- **Legacy Equivalent**:

```
@11_PROCESS=<id>,END;
```

- **Example**:

```
PROC:END 1
```

PROC:ADD

- **Syntax**:

```
PROC:ADD <id>,<step>,<command>
```

- **Description**:

  Adds a step with command and parameters to the process ID.

- **Legacy Equivalent**:

```
@11_PROCESS=<id>,<step>,<command>,<params>;
```

- **Example**:

```
PROC:ADD 1,2,GETVOLT,1
```

PROC:START

- **Syntax**:

```
PROC:START <id>
```

- **Description**:

  Starts the process ID.

- **Legacy Equivalent**:

```
@11_PROCESS=<id>,START;
```

- **Example**:

```
PROC:START 1
```

PROC:STOP

- **Syntax**:

```
PROC:STOP <id>
```

- **Description**:

  Stops the process ID.

- **Legacy Equivalent**:

```
@11_PROCESS=<id>,STOP;
```

- **Example**:

```
PROC:STOP 1
```

PROC:DEL

- **Syntax**:

```
PROC:DEL <id>
```

- **Description**:

  Deletes the process ID.

- **Legacy Equivalent**:

```
@11_PROCESS=<id>,DELETE;
```

- **Example**:

```
PROC:DEL 1
```

1.11.8 Analog IO (Extended)

SOUR:CHAN:VOLT

- **Syntax**:

```
SOUR:CHAN:VOLT <channel>,<voltage>
```

- **Description**:

  Sets the output voltage of the specified channel.

- **Legacy Equivalent**:

```
@11_SETVOLT=<channel>,<voltage>;
```

- **Example**:

```
SOUR:CHAN:VOLT 2,3.300
```

SOUR:CHAN:CURR

- **Syntax**:

```
SOUR:CHAN:CURR <resistorValue>,(@<channel>)
```

- **Description**:

  Sets the current source using a resistor value on the specified channel.

- **Legacy Equivalent**:

```
@11_SETCURR=<channel>,<resistorValue>;
```

- **Example**:

```
SOUR:CHAN:CURR 1200,(@1)
```

MEAS:CHAN:CURR?

- **Syntax**:

```
MEAS:CHAN:CURR? (@<channel>)
```

- **Description**:

  Measures the current on the specified channel.

- **Legacy Equivalent**:

```
@11_GETCUR=<channel>;
```

- **Example**:

```
MEAS:CHAN:CURR? (@1)
0.0123
```

SOUR:CHAN:FREQ

- **Syntax**:

```
SOUR:CHAN:FREQ <frequency>,<dutyCycle>,<voltHigh>,<voltLow>,(@<channel>)
```

- **Description**:

  Configures a PWM output on the specified channel.

- **Legacy Equivalent**:

```
@11_SETFRQ=<channel>,<frequency>,<dutyCycle>,<voltHigh>,<voltLow>;
```

- **Example**:

```
SOUR:CHAN:FREQ 1000,50,5.0,0.0,(@2)
```

MEAS:CHAN:FREQ?

- **Syntax**:

```
MEAS:CHAN:FREQ? (@<channel>)[,<range>[,<window>]]
```

- **Description**:

  Measures the input frequency on the specified channel.

- **Legacy Equivalent**:

```
@11_GETFRQ=<channel>,<range>[,<window>];
```

- **Example**:

```
MEAS:CHAN:FREQ? (@1)
1234.5
```

1.11.9 Ethernet

ETH?

- **Syntax**:

```
ETH?
```

- **Description**:

  Returns the current Ethernet configuration (source, gateway, mask).

- **Legacy Equivalent**:

```
@11_ETH;
```

- **Example**:

```
ETH?
SOURCE,192.168.0.131,GATEWAY,192.168.0.210,MASK,255.255.255.0
```

ETH:SOURCE

- **Syntax**:

```
ETH:SOURCE <ip>
```

- **Description**:

  Sets the source IP address.

- **Legacy Equivalent**:

```
@11_ETH=SOURCE,<ip>;
```

- **Example**:

```
ETH:SOURCE 192.168.0.131
```

ETH:GATEWAY

- **Syntax**:

```
ETH:GATEWAY <ip>
```

- **Description**:

  Sets the gateway IP address.

- **Legacy Equivalent**:

```
@11_ETH=GATEWAY,<ip>;
```

- **Example**:

```
ETH:GATEWAY 192.168.0.210
```

ETH:MASK

- **Syntax**:

```
ETH:MASK <ip>
```

- **Description**:

  Sets the subnet mask address.

- **Legacy Equivalent**:

```
@11_ETH=MASK,<ip>;
```

- **Example**:

```
ETH:MASK 255.255.255.0
```

1.11.10 System Errors

SYST:ERR?

- **Syntax**:

```
SYST:ERR?
```

- **Description**:

  Returns the next error in the system error queue. If no error is present, returns `0`.
- **Error Codes**:
  - **-109**: Missing parameter
  - **-113**: Undefined header
  - **-222**: Execution error
  - **0**: No error
- **Legacy Equivalent**:

```
NONE
```

- **Example**:

```
SYST:ERR?
0
```

1.12 Copyright

1.13 Contact

Tel: +86(21) 6107 5469

Cell: +86 139 1860 5295

Website: www.art-logics.com

Support: support@art-logics.com