The ALB communication protocol allows to take advantage of all the on board resources by means of a set of commands; all the characters received on the serial communication line are interpreted and executed, then an eventual answer can be transmitted to the master control unit. The firmware features also a Setup mode, which allows the user to configure every section of the device.

**WORKING MODE SELECTION**

The firmware of GPC® R/T94 board can manage two different working modes, these are **Setup mode** or **Run mode (ALB)**. The selection of which mode to employ happens during the **Power-ON** phase, by testing the status of jumper **J1 (RUN/DEBUG)**:

\[
\begin{align*}
J1 &= \text{CONNECTED} \quad \rightarrow \text{SETUP Mode} \\
J1 &= \text{NOT CONNECTED} \quad \rightarrow \text{RUN Mode}
\end{align*}
\]

**SETUP MODE**

In **SETUP** mode it is possible to set the initialization parameters, that is the baud rate, the communication mode and the device name. These settings will be stored in the EEPROM, and will make the working configuration in **RUN** mode. To correctly set the initialization parameters please follow the instructions below:

1) Connect jumper **J1** and supply the board.

2) If **SETUP** mode has been recognized then the output OUT0 has been activated (LD1 is ON). At this point the user can set baud rate and communication mode configuring IN0÷IN7 inputs as described in the following tables:

<table>
<thead>
<tr>
<th>BAUD RATE</th>
<th>IN0</th>
<th>IN1</th>
<th>IN2</th>
<th>IN3</th>
<th>IN4</th>
<th>IN5</th>
</tr>
</thead>
<tbody>
<tr>
<td>38400</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>19200</td>
<td>ON</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>9600</td>
<td>OFF</td>
<td>ON</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>4800</td>
<td>OFF</td>
<td>OFF</td>
<td>ON</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>2400</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>ON</td>
<td>OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>1200</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>ON</td>
<td>OFF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>COMMUNICATION MODE</th>
<th>IN6</th>
<th>IN7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point-to-Point</td>
<td>OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>9 bits Master-Slave</td>
<td>ON</td>
<td>OFF</td>
</tr>
</tbody>
</table>

**FIGURE 37: BAUD RATE AND COMMUNICATION MODE TABLES**
To confirm and store the values settings, the user should connect (ON) then disconnect
(OFF) the AUX input. (P3.2).

3) If the previous operation didn't succeed then also output OUT3 is activated (LD4 ON)
and the user must repeat the operation at point 2. If, otherwise, the operation succeeded,
the firmware activates output OUT1 (LD2 ON), to tell that is possible to set the NAME
used by the board for the serial communication. To set the NAME (permitted values
range 128÷255) the same technique described at point 2 should be used, that is, to set an
opportune input configuration on signals IN0÷IN7 then to send an impulse to input AUX
to confirm the data. Here follows a table that shows how to set the name in binary mode:

<table>
<thead>
<tr>
<th>NAME</th>
<th>IN0</th>
<th>IN1</th>
<th>IN2</th>
<th>IN3</th>
<th>IN4</th>
<th>IN5</th>
<th>IN6</th>
<th>IN7</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
<td>OFF</td>
</tr>
<tr>
<td>129</td>
<td>OFF</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
<td>OFF</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>255</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
</tr>
</tbody>
</table>

**FIGURE 38: NAME SETTING TABLE**

4) If the previous operation didn't succeed then also output OUT3 is activated (LD4 ON)
and the user must repeat the operation at point 3. If, otherwise, the operation succeeded,
the firmware stores all the input parameters into the EEPROM and starts an infinite
loop. If this last operation didn't succeed all the OUT0÷OUT3 outputs are activated
(LD1÷LD4 ON), otherwise the firmware states the end of the SETUP procedure setting
the outputs in the following configuration:

- OUT0 = Activated (LD1 ON)
- OUT1 = Deactivated (LD2 OFF)
- OUT2 = Activated (LD3 ON)
- OUT3 = Deactivated (LD4 OFF)

5) Turn off power supply

6) Disconnect jumper J1 then supply the board (RUN mode selected = LD1÷LD4 OFF).

Please note that on optocoupled NPN input is considered actived (ON - LED ON) when the respective
input contact is connected to the ground signal of the optocouplers power supply (GND opto).
RUN MODE

When entering in RUN mode the baud rate and communication protocol parameters stored in EEPROM are verified. If they are not valid (e.g. EEPROM not initialized) the board starts a standby loop; at this point the user may only turn off the device.

EEPROM is NOT initialized by default, the user must initialize it (SETUP mode) before attempting to use the board.

Baud rate and communication protocol for the RUN mode are settable (in SETUP mode), while data format is function of the selected communication mode, as follows:

Point-to-Point Communication: 8 bit, 1 Stop, NO Parity
Master-Slave: 9 bit, 1 Stop, NO Parity

In the following paragraphs the commands recognized in RUN mode are described.

GENERAL COMMANDS

MASTER RESET

Input Sequence:

<table>
<thead>
<tr>
<th>Dec Code</th>
<th>Hex Code</th>
<th>Mnemonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>65</td>
<td>41</td>
<td>A</td>
</tr>
</tbody>
</table>

Upon the reception of this command the firmware restores the initial condition that happens to be at the Power-ON; in detail:

OUT0÷OUT3: They are reset and put into initial condition, then they are set to the logic state 0. Eventual timings are interrupted.
16 bits Counter: It is initialized to 0.

PRESENCE BYTE OUTPUT

Input Sequence:

<table>
<thead>
<tr>
<th>Dec Code</th>
<th>Hex Code</th>
<th>Mnemonic</th>
<th>&lt;NibL VAL&gt;</th>
<th>&lt;NibH VAL&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>89</td>
<td>41</td>
<td>Y</td>
<td>ASCII(NibL VAL)</td>
<td>ASCII(NibH VAL)</td>
</tr>
</tbody>
</table>

The VAL data is stored in the EEPROM area not accessible to the user (0÷31).
This command is ignored in case its sequence contains invalid data.

Example:
If you wish to store the presence byte "65" you will need to send the sequence:
89 1 4.
PRESENCE BYTE INPUT

Input Sequence:

Dec Code: 121
Hex Code: 79
Mnemonic: y

Answer Codes:
A value ranging from 0 to 255 is returned by sending two nibbles in the format seen for the previous command: <NibL VAL>, <NibH VAL>.

FIRMWARE VERSION INPUT

Input Sequence:

Dec Code: 121
Hex Code: 79
Mnemonic: y

Answer Codes:
The two nibbles of the firmware version are returned:

Version X.Y = Y --> <NibL VAL> <NibH VAL>
DIGITAL I/O PORT MANAGEMENT COMMANDS

OUTPUT PORT SET

Input Sequence:

<table>
<thead>
<tr>
<th>Dec Code</th>
<th>Hex Code</th>
<th>Mnemonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>87</td>
<td>57</td>
<td>W SOH ASCII(Dato) NUL</td>
</tr>
</tbody>
</table>

The <Data> byte must be sent according to the following format:

(MSB) 0 0 0 0 OUT3 OUT2 OUT1 OUT0 (LSB)

Where OUTn stands for the logic state, 0 or 1, that the respective output must get. If the sequence contains invalid data the command is ignored.

Example:

If you want to activate the OUT0 and OUT3 outputs you will need to send the following sequence: 87 1 9 0.

INPUT PORT ACQUISITION

Input Sequence:

<table>
<thead>
<tr>
<th>Dec Code</th>
<th>Hex Code</th>
<th>Mnemonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>82</td>
<td>52</td>
<td>R NUL</td>
</tr>
</tbody>
</table>

The serial data read by the input port is returned.

Answer Codes:

The data acquired by the port is returned as nibbles in the following format:

Nibble L:  (MSB) 0 0 0 0 IN3 IN2 IN1 IN0 (LSB)
Nibble H:  0 0 0 0 IN7 IN6 IN5 IN4

Where INn stands for the logic state, 0 or 1, that the respective input have. If the sequence contains invalid data the command is ignored.

Example:

If you want to read the input port, where the 90 (5A Hex) data is present you will need to send the following sequence:

82 0.

Then you will get as answer the following bytes:

10 5.
DIGITAL I/O BIT MANAGEMENT COMMANDS

OUTPUT BIT SET

Input Sequence:

*Dec Code*: 83 1 <Bit>

*Hex Code*: 53 1 <Bit>

*Mnemonic*: S SOH ASCII(Bit)

The output indicated by <Bit> gets the logic state 1; <Bit> can range from 0 to 3. Eventual timings occuring on the output line are interrupted. If the sequence contains invalid data the command is ignored.

Example:

If you want to activate the OUT2 output you will need to send the following sequence:

83 1 2.

TIMED OUTPUT BIT SET

Input Sequence:

*Dec Code*: 115 1 <Bit> <NibL> <NibH>

*Hex Code*: 73 1 <Bit> <NibL> <NibH>

*Mnemonic*: s SOH ASCII(Bit) ASCII(NibL) ASCII(NibH)

The output indicated by <Bit> gets the logic state 1; <Bit> can range from 0 to 3. The selected output holds the logic state 1 for a time determined by the <Nib> bytes, then it returns to the logic state 0. The timing value must range 1÷255 where one unit corresponds to 10 msec and must be sent as nibbles according to the following format:

<table>
<thead>
<tr>
<th>Nibble L: (MSB)</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>Bit3</th>
<th>Bit2</th>
<th>Bit1</th>
<th>Bit0 (LSB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nibble H:</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Bit7</td>
<td>Bit6</td>
<td>Bit5</td>
<td>Bit4</td>
</tr>
</tbody>
</table>

If the sequence contains invalid data the command is ignored.

Example:

If you want to activate the OUT2 output for 500 msec, corresponding to 50 cycles, you will need to send the following sequence:

115 1 2 2 3.
OUTPUT BIT CLEAR

Input Sequence:

<table>
<thead>
<tr>
<th>Dec Code</th>
<th>Hex Code</th>
<th>Mnemonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>67 1</td>
<td>43 1</td>
<td>C SOH</td>
</tr>
</tbody>
</table>

The output indicated by <Bit> gets the logic state 0; <Bit> can range from 0 to 3. Eventual timings occurring on the output line are interrupted. If the sequence contains invalid data the command is ignored.

Example:
If you want to deactivate the OUT2 output, you will need to send the following sequence: 67 1 2.

TIMED OUTPUT BIT CLEAR

Input Sequence:

<table>
<thead>
<tr>
<th>Dec Code</th>
<th>Hex Code</th>
<th>Mnemonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>99 1 &lt;Bit&gt; &lt;NibL&gt; &lt;NibH&gt;</td>
<td>63 1 &lt;Bit&gt; &lt;NibL&gt; &lt;NibH&gt;</td>
<td>c SOH ASCII(Bit) ASCII(NibL) ASCII(NibH)</td>
</tr>
</tbody>
</table>

The output indicated by <Bit> gets the logic state 0; <Bit> can range from 0 to 3. The selected output holds the logic state 0 for a time determined by the <Nib> bytes, then it returns to the logic state 1. The timing value must range 1÷255 where one unit corresponds to 10 msec and must be sent as nibbles according to the following format:

- Nibble L: (MSB) 0 0 0 0 Bit3 Bit2 Bit1 Bit0 (LSB)
- Nibble H: 0 0 0 0 Bit7 Bit6 Bit5 Bit4

If the sequence contains invalid data the command is ignored.

Example:
If you want to deactivate the OUT2 output for 500 msec, corresponding to 50 cycles, you will need to send the following sequence: 99 1 2 2 3.
SQUARE WAVE OUTPUT BIT

Input Sequence:

\[
\begin{array}{ccccccc}
\text{Dec Code}: & 80 & 1 & <\text{Bit}> & <\text{NibL}> & <\text{NibH}> \\
\text{Hex Code}: & 50 & 1 & <\text{Bit}> & <\text{NibL}> & <\text{NibH}> \\
\text{Mnemonic}: & P & \text{SOH} & \text{ASCII(}\text{Bit}) & \text{ASCII(}\text{NibL}) & \text{ASCII(}\text{NibH}) \\
\end{array}
\]

The output indicated by <Bit> outputs a square wave with 50% of Duty Cycle; <Bit> can range from 0 to 3. The the half-period of the signal is determined by the <Nib> bytes. The timing value must range \(1 \div 255\) where one unit corresponds to 10 msec and must be sent as nibbles according to the following format:

- Nibble L: (MSB) 0 0 0 0 Bit3 Bit2 Bit1 Bit0 (LSB)
- Nibble H: 0 0 0 0 Bit7 Bit6 Bit5 Bit4

If the sequence contains invalid data the command is ignored.

Example:
If you want to deactivate the OUT2 output for 200 msec, corresponding to 20 cycles, you will need to send the following sequence: 99 1 2 4 1.

SQUARE WAVE STARTING WITH "1" OUTPUT BIT

Input Sequence:

\[
\begin{array}{ccccccccccc}
\text{Dec Code}: & 112 & 1 & <\text{Bit}> & <\text{NibL}> & <\text{NibH}> & <\text{StaL}> & <\text{StaL}> \\
\text{Hex Code}: & 70 & 1 & <\text{Bit}> & <\text{NibL}> & <\text{NibH}> & <\text{StaH}> & <\text{StaH}> \\
\text{Mnemonic}: & p & \text{SOH} & \text{ASCII(}\text{Bit}) & \text{ASCII(}\text{NibL}) & \text{ASCII(}\text{NibH}) & \text{ASCII(}\text{StaL}) & \text{ASCII(}\text{StaH}) \\
\end{array}
\]

The output starts with "1" and is deactivated after the specified number of cycles.
The output indicated by \(<\text{Bit}>\) outputs a square wave with 50% of Duty Cycle starting by a logical state "1"; \(<\text{Bit}>\) can range from 0 to 3. The half-period of the signal is determined by the \(<\text{Nib}>\) bytes. The timing value must range 1÷255 where one unit corresponds to 10 msec and must be sent as nibbles according to the following format:

\[
\text{Nibble L: (MSB) 0 0 0 0 Bit3 Bit2 Bit1 Bit0 (LSB)}
\]

\[
\text{Nibble H: 0 0 0 0 Bit7 Bit6 Bit5 Bit4}
\]

The permanence of this signal on the selected output is determined by the \(<\text{Sta}>\) byte, which must be sent as nibbles according to the preceding format and must range 1÷255. This byte indicates the number of commutations that must happen on the selected output before it returns steadily to the logic state "0"; the number of commutations is \(<\text{Sta}>+1\) as shown in figure 42.

If the sequence contains invalid data the command is ignored.

Example:
If you want to activate the OUT2 output for 200 msec, corresponding to 20 cycles and making it commutate 10 times, you will need to send the following sequence:
112 1 2 4 1 9 0.

\text{SQUARE WAVE STARTING WITH "0" OUTPUT BIT}

Input Sequence:

\begin{itemize}
\item Dec Code: 119
\item Hex Code: 77
\item Mnemonic: w SOH ASCII(Bit) ASCII(NibL) ASCII(NibH) ASCII(StaL) ASCII(StaH)
\end{itemize}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure43}
\caption{TIMED SQUARE WAVE COMMAND}
\end{figure}

The output indicated by \(<\text{Bit}>\) outputs a square wave with 50% of Duty Cycle starting by a logical state "0"; \(<\text{Bit}>\) can range from 0 to 3. The half-period of the signal is determined by the \(<\text{Nib}>\) bytes. The timing value must range 1÷255 where one unit corresponds to 10 msec and must be sent as nibbles according to the following format:
The permanence of this signal on the selected output is determined by the <Sta> byte, which must be sent as nibbles according to the preceding format and must range \(1 \div 255\). This byte indicates the number of commutations that must happen on the selected output before it returns steadily to the logic state "0"; the number of commutations is \(<\text{Sta}> + 1\) as shown in figure 42. If the sequence contains invalid data the command is ignored.

**Example:**
If you want to deactivate the OUT2 output for 200 msec, corresponding to 20 cycles and making it commutate 10 times, you will need to send the following sequence:
\[119 \quad 1 \quad 2 \quad 4 \quad 1 \quad 9 \quad 0.\]

**INPUT BIT OR AUX INPUT ACQUISITION**

**Input Sequence:**

- **Dec Code:** 114
- **Hex Code:** 72
- **Mnemonic:** r

The logic state, 0 or 1, of the selected <Bit> in the selected <Port>, is returned; <Bit> can range 0÷7 while <Port> can assume the value 0 (INPUT PORT) or 3 (AUX input = Bit 0, other Bit values have no meaning and the command is ignored). If the sequence contains invalid data the command is ignored.

**Example:**
If you want to read the AUX input, you will need to send the following sequence:
\[114 \quad 3 \quad 0\]
The value 0 or 1 will be returned.

**AUX SIGNAL CONFIGURATION ACQUISITION**

**Input Sequence:**

- **Dec Code:** 102
- **Hex Code:** 66
- **Mnemonic:** f

**Answer Codes:**
The AUX signal configuration byte returned by the firmware may assume one of the following values:

\[
\begin{array}{l}
\text{0} & \text{AUX signal is set as an INPUT} \\
\text{2} & \text{AUX signal is used to Trigger the 16 bit Counter}
\end{array}
\]
AUX SIGNAL CONFIGURATION SETTING

Input Sequence:

<table>
<thead>
<tr>
<th>Dec Code</th>
<th>Hex Code</th>
<th>Mnemonic</th>
<th>ASCII(Byte)</th>
</tr>
</thead>
<tbody>
<tr>
<td>85</td>
<td>55</td>
<td>U</td>
<td>U</td>
</tr>
</tbody>
</table>

The AUX signal may be configured as a general purpose INPUT or as a COUNTER according to the value of `<byte>`:

- **0**: AUX signal is set as an INPUT
- **2**: AUX signal is used to Trigger the 16 bit Counter

If the sequence contains invalid data the command is ignored.

**Example:**
If you want to set AUX signal as COUNTER you will need to send the following sequence:

\[
85 \quad 2.
\]
16 BIT COUNTER MANAGEMENT COMMANDS

Here follow the commands to manage the 16 bit counter. Its value is incremented by the commutations of the AUX signal when this is configured to Trigger the counter.

16 BIT COUNTER READ

**Input Sequence:**

- **Dec Code:** 73
- **Hex Code:** 49
- **Mnemonic:** I

This command allows to acquire the current value of the 16 bit counter.

**Answer Codes:**
The sequence returned by the command is made of four bytes showing the 16 bit value currently stored in the counter register; this is sent in nibbles according to the following format:

<table>
<thead>
<tr>
<th>Bit 15</th>
<th>Bit 14</th>
<th>Bit 13</th>
<th>Bit 12</th>
<th>Bit 11</th>
<th>Bit 10</th>
<th>Bit 9</th>
<th>Bit 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 7</td>
<td>Bit 6</td>
<td>Bit 5</td>
<td>Bit 4</td>
<td>Bit 3</td>
<td>Bit 2</td>
<td>Bit 1</td>
<td>Bit 0</td>
</tr>
<tr>
<td>Bit 4</td>
<td>Bit 3</td>
<td>Bit 2</td>
<td>Bit 1</td>
<td>Bit 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bit 8</td>
<td>Bit 7</td>
<td>Bit 6</td>
<td>Bit 5</td>
<td>Bit 4</td>
<td>Bit 3</td>
<td>Bit 2</td>
<td>Bit 1</td>
</tr>
<tr>
<td>Bit 12</td>
<td>Bit 11</td>
<td>Bit 10</td>
<td>Bit 9</td>
<td>Bit 8</td>
<td>Bit 7</td>
<td>Bit 6</td>
<td>Bit 5</td>
</tr>
</tbody>
</table>

When the counter reaches the maximum value, which equals to 65535 (FFFF Hex), the next Trigger impulse will set the counter to 0. If the AUX signal is configured as INPUT this command will always return 0.

**Example:**
If you the counter register contains the value 23055 (5A0F Hex), sending the command **73** will return the following values:

15 0 10 5.

16 BIT COUNTER RESET

**Input Sequence:**

- **Dec Code:** 88 120
- **Hex Code:** 58 78
- **Mnemonic:** X x

Upon the reception of this command the firmware resets the 16 bit counter, which will carry the new value **0**.
MESSAGES MANAGEMENT COMMANDS

LAST MEMORIZABLE MESSAGE ACQUISITION

Input Sequence:
- **Dec Code:** 77
- **Hex Code:** 4D
- **Mnemonic:** M

This command allows the user to know the maximum number of messages that the board can store. This number depends on the memory device installed according to the following table:

<table>
<thead>
<tr>
<th>EEPROM</th>
<th>N.MAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>24c02 (256 Bytes)</td>
<td>23</td>
</tr>
<tr>
<td>24c04 (512 Bytes)</td>
<td>48</td>
</tr>
<tr>
<td>24c08 (1024 Bytes)</td>
<td>99</td>
</tr>
</tbody>
</table>

**FIGURE 44: MAXIMUM NUMBER OF MESSAGES MEMORIZABLE IN EEPROM**

**Answer Codes:**
The number is returned in two nibbles according to the format:

- Nibble L: (MSB) 0 0 0 0 Bit3 Bit2 Bit1 Bit0 (LSB)
- Nibble H: 0 0 0 0 Bit7 Bit6 Bit5 Bit4

READY EEPROM REQUEST

Input Sequence:
- **Dec Code:** 66
- **Hex Code:** 42
- **Mnemonic:** B

By this command the user may ask the firmware if it is ready to manage a new EEPROM message, this command must be sent whenever the user needs to send one of the following messages management commands:

**Answer Codes:**
The firmware returns the following codes:

- **0** EEPROM not ready to manage a new message
- **1** EEPROM ready to manage a new message
STORING A MESSAGE

Input Sequence:

Dec Code:   69  <NibL Mess>  <NibH Mess>
            <NibL Chr.0>  <NibH Chr.0>.....<NibL Chr.0>  <NibH Chr.0>
Hex Code:   45  <NibL Mess>  <NibH Mess>
            <NibL Chr.0>  <NibH Chr.0>.....<NibL Chr.0>  <NibH Chr.0>
Mnemonic:  E  ASCII(NibL Mess)  ASCII(NibH Mess)
            ASCII(NibL Chr.0)  ASCII(NibH Chr.0).....
            ASCII(NibL Chr.9)  ASCII(NibH Chr.9)

The ten characters long message, whose code is indicated by Mess, is stored in EEPROM. The message number must range 0÷N.MAX (see Figure 44 for the value of N.MAX) and must be sent in two nibbles as above indicated. The value of N.MAX may also be acquired by the apposite command. The characters must be sent in two nibbles according to the following format:

\[
\begin{array}{cccc}
\text{Car. x} & \text{Nibble L: (MSB)} & 0 & 0 & 0 & 0 & \text{Bit3} & \text{Bit2} & \text{Bit1} & \text{Bit0} & \text{LSB} \\
\text{Car. x} & \text{Nibble H:} & 0 & 0 & 0 & 0 & \text{Bit7} & \text{Bit6} & \text{Bit5} & \text{Bit4} \\
\end{array}
\]

These byte must range 0÷255 (0÷FF Hex).

Example:
If you want to store the message "ABCDEFGHIJ" (corresponding to the codes: 65, 66, 67, 68, 69, 70, 71, 72, 73, 74) with number 16, you need to send the following sequence:

\[
69  \quad 0  \quad 1  \quad 1  \quad 4  \quad 2  \quad 4  \quad 3  \quad 4  \quad 4  \quad 4  \quad 5  \quad 4  \quad 6  \quad 4  \quad 7  \quad 4  \\
8  \quad 4  \quad 9  \quad 4  \quad 10  \quad 4.
\]

READING A MESSAGE

Input Sequence:

Dec Code:   76  <NibL Mess>  <NibH Mess>
Hex Code:   4C  <NibL Mess>  <NibH Mess>
Mnemonic:  L  ASCII(NibL Mess)  ASCII(NibH Mess)

The ten characters long message is read from the EEPROM and sent on the serial connection. The message number must range 0÷N.MAX (see Figure 44 for the value of N.MAX) and must be sent in two nibbles as above indicated. The value of N.MAX may also be acquired by the apposite command.

Example:
If you want to read the message with number 16 stored in the previous example, you need to send the following sequence: 76  0  1. The answer will be the sequence:

\[
1  \quad 4  \quad 2  \quad 4  \quad 3  \quad 4  \quad 4  \quad 4  \quad 5  \quad 4  \quad 6  \quad 4  \quad 7  \quad 4  \quad 8  \quad 4  \quad 9  \\
4  \quad 10  \quad 4.
\]
RTC AND RAM-RTC MANAGEMENT COMMANDS

CLOCK SETTING

Input Sequence:

Dec Code:  
79 <NibL HOU> <NibH HOU> <NibL MIN> <NibH MIN>  
<NibL SEC> <NibH SEC> <NibL DAY> <NibH DAY>  
<NibL MON> <NibH MON> <NibL YEA> <NibH YEA>  
<NibL DOW> <NibH DOW>

Hex Code:  
4F <NibL HOU> <NibH HOU> <NibL MIN> <NibH MIN>  
<NibL SEC> <NibH SEC> <NibL DAY> <NibH DAY>  
<NibL MON> <NibH MON> <NibL YEA> <NibH YEA>  
<NibL DOW> <NibH DOW>

Mnemonic:  
O ASCII(NibL HOU) ASCII(NibH HOU)  
ASCII(NibL MIN) ASCII(NibH MIN)  
ASCII(NibL SEC) ASCII(NibH SEC)  
ASCII(NibL DAY) ASCII(NibH DAY)  
ASCII(NibL MON) ASCII(NibH MON)  
ASCII(NibL YEA) ASCII(NibH YEA)  
ASCII(NibL DOW) ASCII(NibH DOW)

<table>
<thead>
<tr>
<th>BYTE</th>
<th>RANGE</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOU</td>
<td>0 ... 23</td>
<td>HOURS</td>
</tr>
<tr>
<td>MIN</td>
<td>0 ... 59</td>
<td>MINUTES</td>
</tr>
<tr>
<td>SEC</td>
<td>0 ... 59</td>
<td>SECONDS</td>
</tr>
<tr>
<td>DAY</td>
<td>1 ... 31</td>
<td>DAY</td>
</tr>
<tr>
<td>MON</td>
<td>1 ... 12</td>
<td>MONTH</td>
</tr>
<tr>
<td>YEA</td>
<td>0 ... 99</td>
<td>YEAR</td>
</tr>
<tr>
<td>DOW</td>
<td>0 ... 6</td>
<td>Day of week</td>
</tr>
</tbody>
</table>

FiguRe 45: RTC INITIALIZATION BYTES VALIDITY RANGE

If the sequence contains invalid data the command is ignored.

Example:
If you wish to set the RTC as: Monday May 11th 1998 12:30:40 you will need to send:
79 12 0 14 1 8 2 11 0 5 0 2 6 1 0.
CLOCK READ

Input Sequence:

Dec Code: 111
Hex Code: 6F
Mnemonic: o

Answer Codes:
The RTC bytes are sent in nibbles according to the format seen for the previous command:

\[
\begin{align*}
\text{<NibL HOU>} & \quad \text{<NibH HOU>} & \quad \text{<NibL MIN>} & \quad \text{<NibH MIN>} \\
\text{<NibL SEC>} & \quad \text{<NibH SEC>} & \quad \text{<NibL DAY>} & \quad \text{<NibH DAY>} \\
\text{<NibL MON>} & \quad \text{<NibH MON>} & \quad \text{<NibL YEA>} & \quad \text{<NibH YEA>} \\
\text{<NibL DOW>} & \quad \text{<NibH DOW>}
\end{align*}
\]

If the sequence contains invalid data the command is ignored.

RTC RAM WRITE

Input Sequence:

Dec Code: 71 <NibL IND> <NibH IND> <NibL VAL> <NibH VAL>
Hex Code: 47 <NibL IND> <NibH IND> <NibL VAL> <NibH VAL>
Mnemonic: G ASCII(NibL IND) ASCII(NibH IND)
ASCII(NibL VAL) ASCII(NibH VAL)

The VAL data (0÷255) is stored in the RTC RAM at the address IND (32÷255).
If the sequence contains invalid data the command is ignored.

Example:
If you wish to store data "65" at the address "100" you will need to send the sequence:
71 4 6 1 4.

RTC RAM READ

Input Sequence:

Dec Code: 103 <NibL IND> <NibH IND>
Hex Code: 67 <NibL IND> <NibH IND>
Mnemonic: g ASCII(NibL IND) ASCII(NibH IND)

The data stored in the RTC RAM at the address IND (32÷255) is read.
If the sequence contains invalid data the command is ignored.

Answer Codes:
The data (0÷255) is sent in nibbles according to the format seen for the previous command:

<NibL VAL> <NibH VAL>.
<table>
<thead>
<tr>
<th>COMMAND</th>
<th>CODE</th>
<th>HEX</th>
<th>MNEMONIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master Reset</td>
<td>65 97</td>
<td>41 61</td>
<td>A a</td>
</tr>
<tr>
<td>Presence ByteWrite</td>
<td>89</td>
<td>59</td>
<td>Y ASCII(val.L) ASCII(val.H)</td>
</tr>
<tr>
<td>Presence Byte Read</td>
<td>121</td>
<td>79</td>
<td>y</td>
</tr>
<tr>
<td>Firmware Version Read</td>
<td>86</td>
<td>56</td>
<td>V</td>
</tr>
<tr>
<td>OUTPUT Port Setting</td>
<td>87 1</td>
<td>57 1</td>
<td>W SOH ASCII(Dato) NUL</td>
</tr>
<tr>
<td>INPUT Port Acquisition</td>
<td>82 0</td>
<td>52 0</td>
<td>R NUL</td>
</tr>
<tr>
<td>SET Port.Bit</td>
<td>83 1</td>
<td>53 1</td>
<td>S SOH ASCII(bit)</td>
</tr>
<tr>
<td>CLEAR Port.Bit</td>
<td>67 1</td>
<td>43 1</td>
<td>C SOH ASCII(bit)</td>
</tr>
<tr>
<td>Timed SET Port.Bit</td>
<td>115 1</td>
<td>73 1</td>
<td>s SOH ASCII(bit) ASCII(nib.L) ASCII(nib.H)</td>
</tr>
<tr>
<td>Timed CLEAR Port.Bit</td>
<td>99 1</td>
<td>63 1</td>
<td>c SOH ASCII(bit) ASCII(nib.L) ASCII(nib.H)</td>
</tr>
<tr>
<td>Square Wave Port.Bit</td>
<td>80 1</td>
<td>50 1</td>
<td>P SOH ASCII(bit) ASCII(nib.L) ASCII(nib.H)</td>
</tr>
<tr>
<td>Square Wave starting with &quot;1&quot; Port.Bit</td>
<td>112 1</td>
<td>70 1</td>
<td>p SOH ASCII(bit) ASCII(nib.L) ASCII(nib.H) ASCII(sta.L) ASCII(sta.H)</td>
</tr>
<tr>
<td>Square Wave starting with &quot;0&quot; Port.Bit</td>
<td>119 1</td>
<td>77 1</td>
<td>w SOH ASCII(bit) ASCII(nib.L) ASCII(nib.H) ASCII(sta.L) ASCII(sta.H)</td>
</tr>
<tr>
<td>Port.Bit Acquisition</td>
<td>114 port bit</td>
<td>72 port bit</td>
<td>r ASCII(port) ASCII(bit)</td>
</tr>
<tr>
<td>AUX Signal Config Read</td>
<td>102</td>
<td>66</td>
<td>f</td>
</tr>
<tr>
<td>AUX Signal Config Write</td>
<td>85 byte</td>
<td>55 byte</td>
<td>U ASCII(byte)</td>
</tr>
</tbody>
</table>

**Figure 46: Commands Table 1**
<table>
<thead>
<tr>
<th>Command</th>
<th>Code</th>
<th>HEX</th>
<th>Mnemonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset Counter</td>
<td>88 120</td>
<td>58 78</td>
<td>X x</td>
</tr>
<tr>
<td>Counter Read</td>
<td>73</td>
<td>49</td>
<td>I</td>
</tr>
<tr>
<td>Ready EEPROM Request</td>
<td>66</td>
<td>42</td>
<td>B</td>
</tr>
<tr>
<td>Read last message number</td>
<td>77</td>
<td>4D</td>
<td>M</td>
</tr>
<tr>
<td>Read RTC</td>
<td>111</td>
<td>6F</td>
<td>o</td>
</tr>
<tr>
<td>Write RAM RTC</td>
<td>71 nib.L0 nib.H0 nib.L1 nib.H1</td>
<td>47 nib.L0 nib.H0 nib.L1 nib.H1</td>
<td>G ASCII(nib.L0) ASCII(nib.H0) ASCII(nib.L1) ASCII(nib.H1)</td>
</tr>
</tbody>
</table>

**Figure 47: Commands table 2**
9 BITS MASTER-SLAVE COMMUNICATION MODE

The Master-Slave communication takes advantage of the 9 bits mode, this means that a ninth bit is used to distinguish between a call from a "Master" device to one of the "Slave" structures and a mere communication of data between the Master and the selected Slave.

When the ninth bit is set to 1, the data byte must contain the name, or ID code, of the new target device, while if the ninth bit is set to 0 it is possible to send or receive informations from the selected target device. If the communication is running under ALB protocol the ID code must be the byte set in **SETUP mode (NAME)**. When this byte is received by a device (with the ninth bit set to 1) it recognizes itself and starts to wait for a string containing data or commands (with the ninth bit set to 0); the string must be maximum 24 bytes long.

It can contain only one command which requires to return an answer code through the serial line, more commands of this kind will be ignored.

The delay between two consecutive characters must be lower than **Time-Out**, because otherwise the string is considered terminated and the answering phase starts.

Here follows the list of Time-Outs related to the Baud Rate:

<table>
<thead>
<tr>
<th>Baud Rate</th>
<th>Time-Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>38400 Baud</td>
<td>550 µsec</td>
</tr>
<tr>
<td>19200 Baud</td>
<td>990 µsec</td>
</tr>
<tr>
<td>9600 Baud</td>
<td>1.54 msec</td>
</tr>
<tr>
<td>4800 Baud</td>
<td>3.08 msec</td>
</tr>
<tr>
<td>2400 Baud</td>
<td>6.105 msec</td>
</tr>
<tr>
<td>1200 Baud</td>
<td>12.1 msec</td>
</tr>
</tbody>
</table>

When the Time-Out happens, the answer sequence begins; this is made of a byte containing the presence code 6 (6 Hex), or a data sequence requested by a read command sent during the previous call.

**Example:**
If a string containing the Port read command is sent, the answer to that call will be the presence code, while the answer to the next call will be the data acquired by the Port sent in the previous request.

After having sent the last character of the string the user will have to wait for a time:

"One char transmission time" + Time-Out

before receiving the first char of the answer sequence.

**Example:**
At 38.4 KBAud, After having sent the last character of the string the user will have to wait for about 810 µsec before to receive the first char of the answer sequence.
**FIGURE 48: GPC® R/T94 POSSIBLE CONNECTIONS**

- **4 OUTPUT LINES**
  - RELAYS or TRANSISTORS

- **POWER SUPPLY**
  - +10÷40 Vdc
  - OR
  - +10÷24 Vac
  - +Vopto (+24 Vdc)

- **1 A/D LINE**
  - Resolution = 11 bits
  - Conversion time = 60 ms

- **9 INPUT LINES**
  - OPTO COUPLED

- **DIN 46277-1 and DIN 46277-2 OMEGA RAILS**

- **1 Hardware Serial Line**
  - TTL, RS-232, RS 422, RS 485, Current Loop

- **PC like or Macintosh**
- **PLC**
- **QTP series**
- **GPC® R/T94**
NOTES:

1) Between two calls the user should wait for a time which depends on the number of commands sent and the kind of operations their execution involves. A string of datas or commands sent by Master must always contain complete sequences. If one of these is incomplete it may be ignored, and so the next sequence, even if complete.

2) If the Master unit cannot manage 9 bits communication, it is possible to simulate the ninth bit by programming the parity bit, before sending one byte, according to the following scheme:

   **The byte to send has an EVEN number of bits "1"**
   - If bit 9 has to be 1 -> Set parity ODD
   - If bit 9 has to be 0 -> Set parity EVEN

   **The byte to send has an ODD number of bits "1"**
   - If bit 9 has to be 1 -> Set parity EVEN
   - If bit 9 has to be 0 -> Set parity ODD