ALB (ABACO® LINK BUS) SOFTWARE DESCRIPTION

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 beretransmitted 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 testin the status of jumper **J1 (RUN/DEBUG)**:

J1	=	**CONNECTED**	--> SETUP Mode
J1	=	**NOT CONNECTED**	--> RUN Mode

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:

BAUD RATE	IN0	IN1	IN2	IN3	IN4	IN5
38400	OFF	OFF	OFF	OFF	OFF	OFF
19200	ON	OFF	OFF	OFF	OFF	OFF
9600	OFF	ON	OFF	OFF	OFF	OFF
4800	OFF	OFF	ON	OFF	OFF	OFF
2400	OFF	OFF	OFF	ON	OFF	OFF
1200	OFF	OFF	OFF	OFF	ON	OFF

COMMUNICATION MODE	IN6	IN7
Point-to-Point	OFF	OFF
9 bits Master-Slave	ON	OFF

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 ouptut OUT3 is activated (LD4 ON) and the user must repeat the operation at point 2. If, otherwise, the operation secceeded, the firmware activates output OUT1 (LD2 ON), to tell that it 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:

NAME	IN0	IN1	IN2	IN3	IN4	IN5	IN6	IN7
128	ON	ON	ON	ON	ON	ON	ON	OFF
129	OFF	ON	ON	ON	ON	ON	ON	OFF
...	...	...	...	...	...	...	...	...
255	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF

FIGURE 38: NAME SETTING TABLE

4) If the previous operation didn't succeed then also ouptut OUT3 is activated (LD4 ON) and the user must repeat the operation at point 3. If, otherwise, the operation secceeded, 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 grond 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 stand-by loop; at this point the user may only turn off the device.

EEPROM is NOT initialized by default, the 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 mde, 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:

Dec Code:	**65**	**97**
Hex Code:	**41**	**61**
Mnemonic:	**A**	**a**

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:

Dec Code:	**89**	**<NibL VAL>**	**<NibH VAL>**
Hex Code:	**41**	**<NibL VAL>**	**<NibH VAL>**
Mnemonic:	**Y**	**ASCII(NibL VAL)**	**ASCII(NibH VAL)**

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:

Dec Code:	**87**	**1**	**<Data>**	**0**
Hex Code:	**57**	**1**	**<Data>**	**0**
Mnemonic:	**W**	**SOH**	**ASCII(Dato)**	**NUL**

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:

Dec Code:	**82**	**0**
Hex Code:	**52**	**0**
Mnemonic:	**R**	**NUL**

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 occourring 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)**

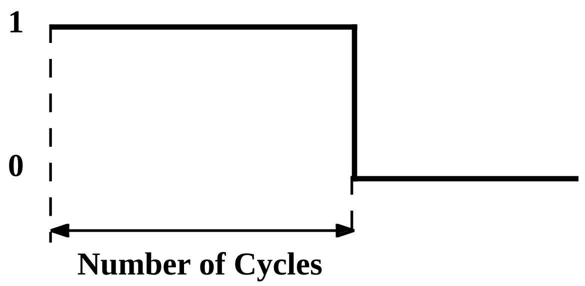

FIGURE 39: TIMED SET COMMAND

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:

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 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:

Dec Code:	**67**	**1**
Hex Code:	**43**	**1**
Mnemonic:	**C**	**SOH**

The output indicated by **<Bit>** gets the logic state 0; **<Bit>** can range from 0 to 3.
Eventual timings occourring 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:

Dec Code:	**99**	**1**	**<Bit>**	**<NibL>**	**<NibH>**
Hex Code:	**63**	**1**	**<Bit>**	**<NibL>**	**<NibH>**
Mnemonic:	**c**	**SOH**	**ASCII(Bit)**	**ASCII(NibL)**	**ASCII(NibH)**

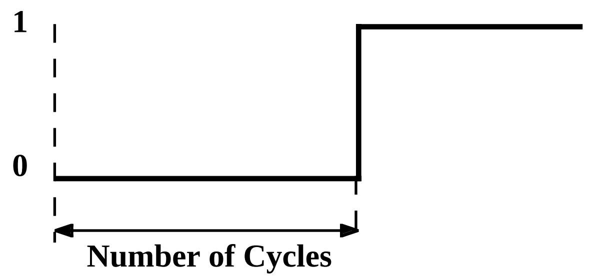

FIGURE 40: TIMED CLEAR COMMAND

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:

Dec Code:	**80**	**1**	**<Bit>**	**<NibL>**	**<NibH>**
Hex Code:	**50**	**1**	**<Bit>**	**<NibL>**	**<NibH>**
Mnemonic:	**P**	**SOH**	**ASCII(Bit)**	**ASCII(NibL)**	**ASCII(NibH)**

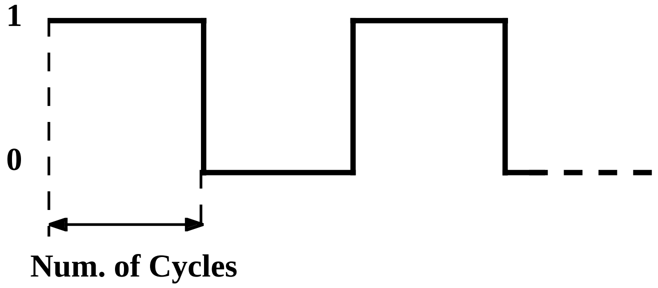

FIGURE 41: SQUARE WAVE COMMAND

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÷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:

Dec Code:	**112**	**1**	**<Bit>**	**<NibL>**	**<NibH>**	**<StaL>**	**<StaL>**
Hex Code:	**70**	**1**	**<Bit>**	**<NibL>**	**<NibH>**	**<StaH>**	**<StaH>**
Mnemonic:	**p**	**SOH**	**ASCII(Bit)**	**ASCII(NibL)**	**ASCII(NibH)**	**ASCII(StaL)**	**ASCII(StaH)**

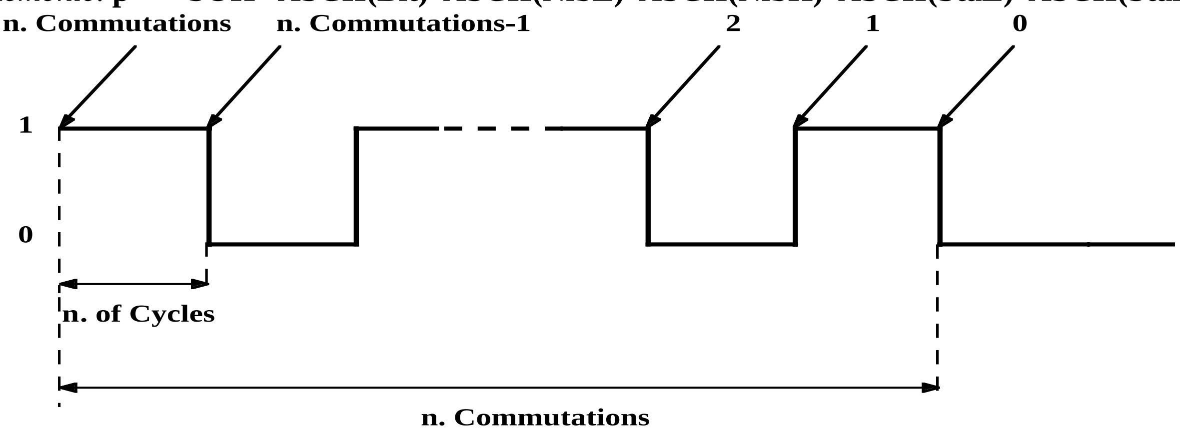

FIGURE 42: TIMED SQUARE WAVE COMMAND

The output indicated by **<Bit>** outputs a square wave with 50% of Duty Cycle **starting by a logical state "1"**; **<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÷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**	

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÷255**. This byte indicates the number of commutations that must happens on the selected output before it returns steadly to the logic state "0"; the number of commutations is **<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**.**

SQUARE WAVE STARTING WITH "0" OUTPUT BIT

Input Sequence:

Dec Code:	**119**	**1**	**<Bit>**	**<NibL>**	**<NibH>**	**<StaL>**	**<StaL>**
Hex Code:	**77**	**1**	**<Bit>**	**<NibL>**	**<NibH>**	**<StaH>**	**<StaH>**
Mnemonic:	**w**	**SOH**	**ASCII(Bit)**	**ASCII(NibL)**	**ASCII(NibH)**	**ASCII(StaL)**	**ASCII(StaH)**

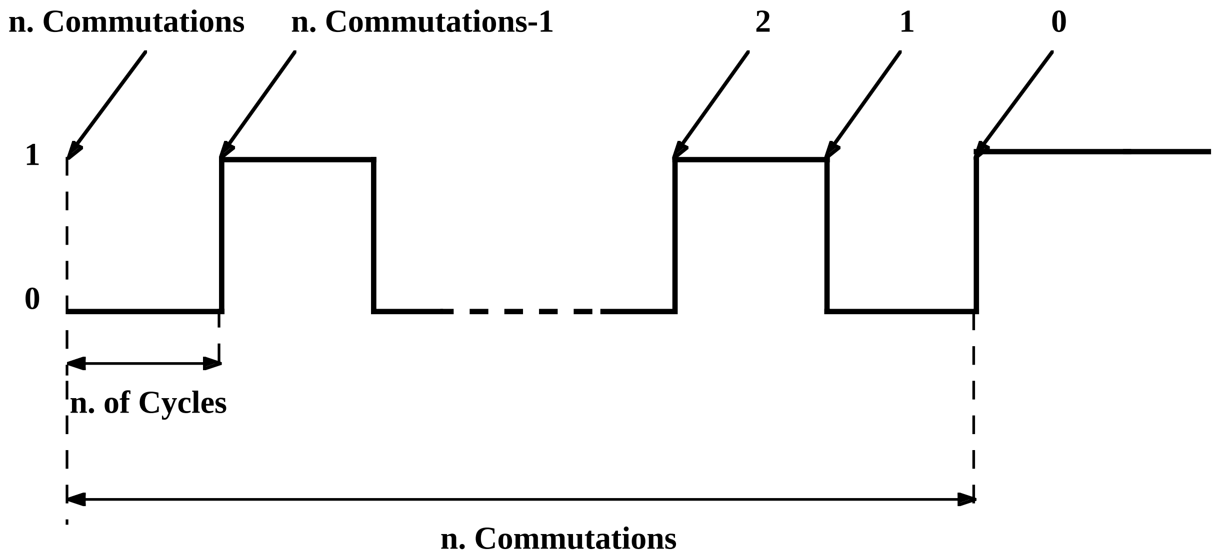

FIGURE 43: TIMED SQUARE WAVE COMMAND

The output indicated by **<Bit>** outputs a square wave with 50% of Duty Cycle **starting by a logical state "0"**; **<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÷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**	

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÷255**. This byte indicates the number of commutations that must happens on the selected output before it returns steadly to the logic state "0"; the number of commutations is **<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 1 2 4 1 9 0.

INPUT BIT OR AUX INPUT ACQUISITION

Input Sequence:

Dec Code:	**114**	**<Port>**	**<Bit>**
Hex Code:	**72**	**<Port>**	**<Bit>**
Mnemonic:	**r**	**ASCII(Port)**	**ASCII(Bit)**

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 3 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:

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

AUX SIGNAL CONFIGURATION SETTING

Input Sequence:

Dec Code:	**85**	**<Byte>**
Hex Code:	**55**	**<Byte>**
Mnemonic:	**U**	**ASCII(Byte)**

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 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:

Counter	(bit 0÷3)	:	(MSB)	0	0	0	0	**Bit3**	**Bit2**	**Bit1**	**Bit0**	(LSB)
	(bit 4÷7)	:		0	0	0	0	**Bit7**	**Bit6**	**Bit5**	**Bit4**	
	(bit 8÷11)	:		0	0	0	0	**Bit11**	**Bit10**	**Bit9**	**Bit8**	
	(bit 12÷15)	:		0	0	0	0	**Bit15**	**Bit14**	**Bit13**	**Bit12**	

When the counter reaches the maximum value, which equals to 65535 (FFFF Hex), the next Trigger impulse will set the counter to 0. If th eAUX signal is configures 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 memoty device installed according to the following table:

EEPROM	N.MAX
24c02 (256 Bytes)	23
24c04 (512 Bytes)	48
24c08 (1024 Bytes)	99

FIGURE 44: MAXIMUM NUMBER OF MESSAGES MEMORIZABLE IN EEPROM

Answer Codes:
The nuber 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 firware 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:

Car. x	Nibble L:	(MSB)	0	0	0	0	**Bit3**	**Bit2**	**Bit1**	**Bit0**	(LSB)
Car. x	Nibble H:		0	0	0	0	**Bit7**	**Bit6**	**Bit5**	**Bit4**	

These byte must range **0÷255** (**0÷FF** Hex).
If the sequence contains invalid data the command is ignored.

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 **0 1 1 4 2 4 3 4 4 4 5 4 6 4 7 4**
8 4 9 4 10 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.
If the sequence contains invalid data the command is ignored.

Answer Codes:
The ten characters are returned in nibbles according to the format seen for the previous command.

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 4 2 4 3 4 4 4 5 4 6 4 7 4 8 4 9
4 10 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 YEA)	**ASCII(NibH YEA)**		
		ASCII(NibL DOW)	**ASCII(NibH DOW)**		

BYTE	RANGE	MEANING
HOU	0 ... 23	HOURS
MIN	0 ... 59	MINUTES
SEC	0 ... 59	SECONDS
DAY	1 ... 31	DAY
MON	1... 12	MONTH
YEA	0 ... 99	YEAR
DOW	0 ... 6	Day of week : 0 -> SUNDAY ..... 6 -> SATURDAY

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:

<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>**		

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>.

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

FIGURE 46: COMMANDS TABLE 1

COMMAND	CODE	HEX	MNEMONIC
Reset Counter	88 120	58 78	X x
Counter Read	73	49	I
Ready EEPROM Request	66	42	B
Read last message number	77	4D	M
Store Message	69 nib.L nib.H nib.L0 nib.H0 .... nib.L9 nib.H9	45 nib.L nib.H nib.L0 nib.H0 .... nib.L9 nib.H9	E ASCII(nib.L) ASCII(nib.H) ASCII(nib.L0) ASCII(nib.H0) .... ASCII(nib.L9) ASCII(nib.H9)
Read Message	76 nib.L nib.H	4C nib.L nib.H	L ASCII(nib.L) ASCII(nib.H)
RTC Setting	79 nib.L0 nib.H0 .... nib.L6 nib.H6	4F nib.L0 nib.H0 .... nib.L6 nib.H6	O ASCII(nib.L0) ASCII(nib.H0) .... ASCII(nib.L6) ASCII(nib.H6)
Read RTC	111	6F	o
Write RAM RTC	71 nib.L0 nib.H0 nib.L1 nib.H1	47 nib.L0 nib.H0 nib.L1 nib.H1	G ASCII(nib.L0) ASCII(nib.H0) ASCII(nib.L1) ASCII(nib.H1)
Read RAM RTC	103 nib.L nib.H	67 nib.L nib.H	g ASCII(nib.L) ASCII(nib.H)

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 nineth bit is used to distinguish between a call from a "**Master**" device to one of the "**Slave**" structures and a mere communication of datas between the Master and the selected Slave.
When the nineth bit is set to 1, the data byte must contain the name, or ID code, of the new target device, while if the nineth 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 nineth bit set to 1**) it recognizes itself and starts to wait for a string containing dats or commands (**with the nineth bit set to 0**); the string must be maximun **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:

Baud Rate	**Time-Out**
38400 Baud	550 μsec
19200 Baud	990 μsec
9600 Baud	1.54 msec
4800 Baud	3.08 msec
2400 Baud	6.105 msec
1200 Baud	12.1 msec

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.

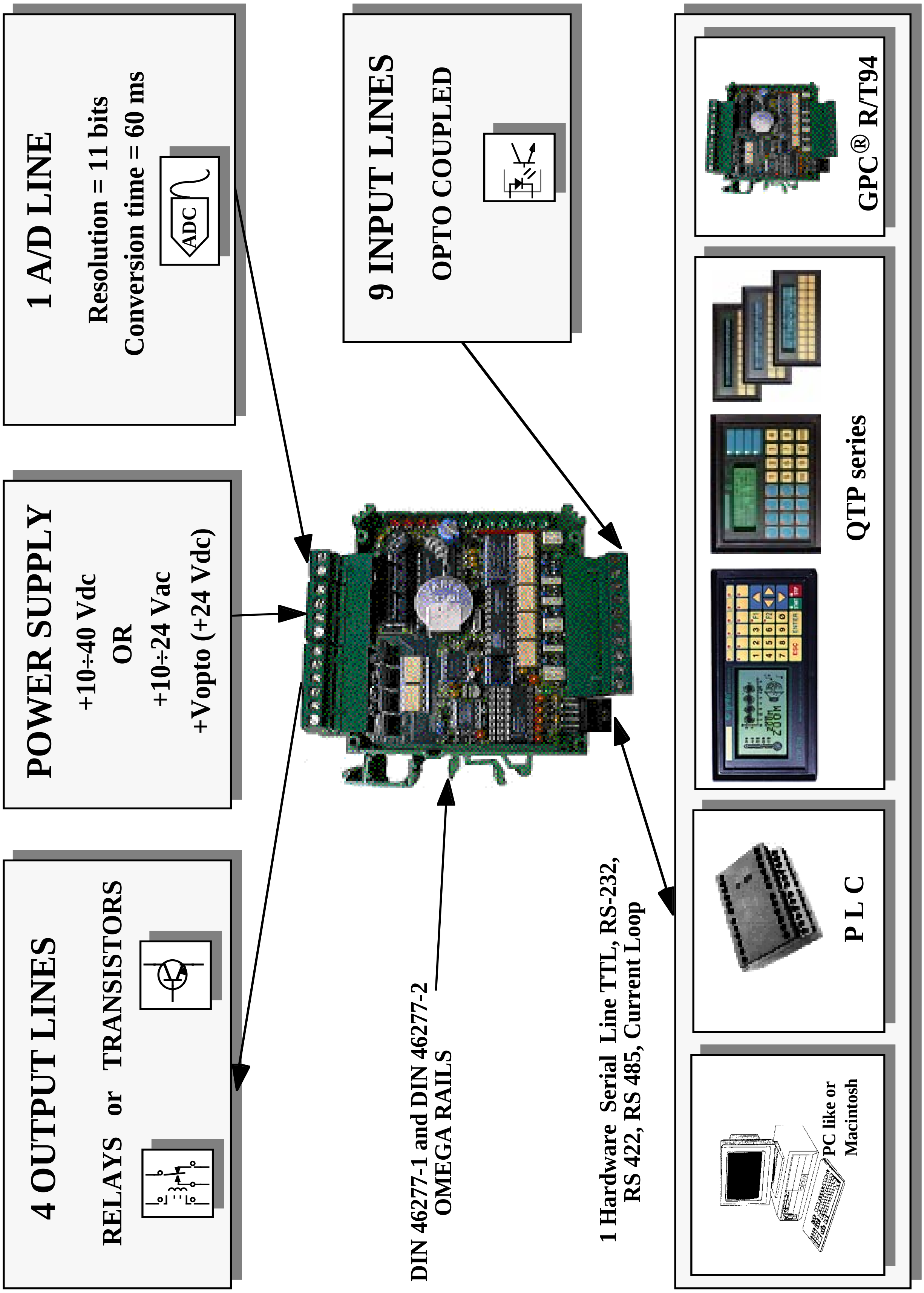

FIGURE 48: GPC® R/T94 POSSIBLE CONNECTIONS DIAGRAM

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 nineth bit by programming the parity bit, before sending one byte, according t 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*