

BASIC 553

rel. 3.1 and later

INTRODUCTION

BASIC 52 is a powerful software tool developed by **grifo®**, capable of managing a BASIC high level programming of all Intel 51 family based cards. It is a "romated" software that generates "romable" software in an easy to use environment; it also reduces the necessity of external hardware (in circuit emulator, EPROM programmer, etc.) and at the same time it speeds up the debugging phase of the User application program.

BASIC 52 is referred to generic software tools, but each card has a specific version of software associated to their hardware features; so for each card the name **BASIC 52** become BASIC followed by the card final name.

This documentation must be read as "additional up-grade" **BASIC 52** manual.

BASIC 52 FOR GPC® 553 = BASIC 553

Here follows a summary of the differences between original MCS BASIC 52 and **BASIC 553**:

Removed commands

LIST#
FPROG
FPROG1
FPROG2
FPROG3
FPROG4
FPROG5
FPROG6

Removed instructions

BAUD
PRINT#
PH0.#
PH1.#
PWM

Removed operators

None

Added commands

ERASE

->

Deletes EEPROM content removing all the application program saved in with command PROG, PROG1, ..., PROG6.

Function

Added instructions

GES_RTC
ALARM
A_D
SET_PWM
BY_EE
BL_EE

->
->
->
->
->
->

Manages RTC initialize and read.
Manages the several RTC alert modes.
Manages the on board A/D acquisitions.
Manages CPU PWM lines.
Performs a byte read or write operation on serial EEPROM.
Performs a data block read or write operation on serial EEPROM.

Function

Added instructions

Function

RW_SFR	->	Performs special function registers (SFR) read or write operation.
COM2	->	Manages INPUT on the software serial line.
PRINT@	->	Manages OUTPUT on the software serial line.
DISPLAY	->	Selects and initializes a display.
KEYB	->	Manages all the keyboard operations.
Output console redirection	->	Manages LCD or fluorescent display through BASIC high level instructions.

Added operators

Function

None	->	-
------	----	---

AUTORUN AND DEBUG MODE

For executing an application program in AUTORUN mode (automatic start after power on or reset), the User must move the DIP 8 on the dip switch **DSW1** to **ON**. While if **BASIC 553** must be executed when an AUTORUN condition is present, the User must the DIP 8 on the dip switch **DSW1** to **OFF**. For summarizing:

DSW1.8 OFF: DEBUG mode.
DSW1.8 ON: RUN mode.

DEBUG mode is really interesting when, for example, a working system must be changed or updated: in this condition the User can simply connect target card to P.C., selecting the DEBUG mode and with **BASIC 553**, loading, testing and saving a new application program and then set again RUN mode. No hardware must be changed on target card, being possible to make these operations directly "on the field". For setting a program in AUTORUN mode follow these steps:

- Select DEBUG mode (DSW1.8 OFF).
- Type: ERASE <enter>.
- Load the program in memory.
- Type: PROG <enter>
- Type: PROG4 <enter>

ADDITIONAL INSTRUCTION SYNTAX DESCRIPTION

To **BASIC 553** have been added some new procedures which allow management of **GPC® 553** on board hardware. Some external SRAM locations are used for parameters interchange between application program and added procedures and for the same procedures own requirements. This SRAM area is located in **07E00H-07FFF** addresses range. Before writing an application program, that uses the additional procedures, the User must absolutely execute the command: **MTOP=07DFFH** to avoid overlap between interchange area and **BASIC 553** work SRAM area.

The User must not change internal ram locations 018H-021H because implement the new tokens. The next pages describes additional procedures with information about parameters and execution.

A/D CONVERSION MANAGEMENT PROCEDURE

Syntax: A_D <expr>

Procedure description:

It performs an A/D conversion of one of the eight **GPC® 553** analog inputs. The conversion is made on the requested channel, and the result (with 10 bits resolution = 0÷1023), is returned to main program.

Parameters description:

<expr> --> Channel number (0÷7).

Procedure output description (values that must extract through POP ... instruction):

OUTPUT: 0÷1023 --> Conversion result value.
65535 --> Parameter error.

Example:

```
0 REM ***** Example of A/D conversion.
10 A_D 3 : REM Conversion on channel 3.
20 POP A : REM It reads the conversion value.
30 IF A<>65535 THEN PRINT A : REM It prints the value.
```

CPU PWM LINES MANAGEMENT PROCEDURE

Syntax: SET_PWM <expr1>, <expr2>, <expr3>

Procedure description:

It generates PWM signals on CPU line 0 and 1. For calculating the PWM signal frequency and duty cycle the following formulas can be used:

$$F_{\text{PWM}} = 22118400 / (2 \times (1 + \text{PWMP}) \times 255).$$

$$\text{low/high ration of /PWMn} = (\text{PWMn} / 255) - \text{PWMn}$$

Parameters description:

<expr1> --> PWM line selection (0÷1).

<expr2> --> Frequency (169÷43369 Hz with a 22118400 Hz crystal).

<expr3> --> Duty Cycle (0÷100%).

If <expr2> and <expr3> are both set to 0 the PWM line is set and mantained at "0" logic value.

If <expr2> and <expr3> are both set to 1 the PWM line is set and mantained at "1" logic value.

Procedure output description (values that must extract through POP ... instruction):

OUTPUT: None.

Examples:

```
0 REM ***** Demo program for generating PWM signal on CPU line 0.
```

```
10 SET_PWM 0, 180, 50 : REM It sets PWM0; 180 Hz; 50%.
```

```
0 REM ***** Demo program for generating PWM signal on CPU line 1.
```

```
10 SET_PWM 1, 180, 50 : REM It sets PWM1; 180 Hz; 50%.
```

```
0 REM ***** Demo program for setting PWM line 0 to "0" logic value.
```

```
10 SET_PWM 0, 0, 0 : REM It sets PWM0 to "0";
```

```
0 REM ***** Demo program for setting PWM line 0 to "1" logic value.
```

```
10 SET_PWM 0, 1, 1 : REM It sets PWM0 to "1";
```

```
0 REM ***** Demo program for setting PWM line 1 to "0" logic value.
```

```
10 SET_PWM 1, 0, 0 : REM It sets PWM1 to "0";
```

```
0 REM ***** Demo program for setting PWM line 1 to "1" logic value.
```

```
10 SET_PWM 1, 1, 1 : REM It sets PWM1 to "1";
```

RTC MANAGEMENT PROCEDURE

Syntax: GES_RTC <expr1>, <expr2>, <expr3>, <expr4>, <expr5>, <expr6>, <expr7>, <expr8>

Procedure description:

Initializes the Real Time Clock or returns date or time. Please remark that also when reading data the User must supply all the parameters even if they are not significant.

Parametrs description:

<expr1> --> 0 = Read hours, minutes, seconds.
 --> 1 = Read day of week, day, month, year.
 --> 2 = Clock initialization.
 <expr2> --> New value for hours (0÷23).
 <expr3> --> New value for minutes (0÷59).
 <expr4> --> New value for seconds (0÷59).
 <expr5> --> New value for day of week (0÷6).
 <expr6> --> New value for day of month (1÷31).
 <expr7> --> New value for month (1÷12).
 <expr8> --> New value for year (0÷3).

Procedure output description (values that must extract through POP ... instruction):

OUTPUT INITIALIZATION: 0 --> Initialization OK.
 <>0 --> The device is not recognized.

OUTPUT READING: 0÷255 --> Read OK.
 >255 --> The device is not recognized.

Example:

```

0   REM ***** Demo program for setting and reading the on board clock.
10  GES_RTC 2,23,59,30,6,31,12,2 : REM Clock initialization.
20  POP A : REM Read the result of the operation
30  IF A<>0 THEN PRINT "INITIALIZATION ERROR" : END
40  GES_RTC 1,0,0,0,0,0,0,0 : REM Date reading.
50  POP SET : POP GIO : POP MES : POP ANN
60  IF (SET>255.OR.GIO>255.OR.MES>255.OR.ANN>255) THEN 120
70  ? SET, : ? CHR(45), : ? GIO, : ? CHR(45), : ? MES, : ? CHR(45), : ? ANN+1900,
80  GES_RTC 0,0,0,0,0,0,0,0 : REM Time reading.
90  POP A : POP B : POP C
100 IF (A>255.OR.B>255.OR.C>255) THEN 120
110 ? " " : ? A, : ? CHR(58), : ? B, : ? CHR(58), : ? C, : ? CHR(13), : GOTO 40
120 PRINT "CLOCK READING ERROR." : END

```

SERIAL EEPROM BYTE READ/WRITE PROCEDURE

Syntax: BY_EE <expr1>, <expr2>, <expr3>

Procedure description:

It performs a byte read or write operation on serial EEPROM (IC16) or on PCF 8583 (IC17) at a specified address.

The user must remember that in read procedure the <expr3> parameter must be given even if it has no meaning.

Parameters description:

<expr1> --> 0 = Reading of a byte.
 --> 1 = Writing of a byte.
<expr2> --> Location address (0÷<last device address>).
<expr3> --> Byte to write (0÷255).

Serial EEPROM and SRAM-RTC addresses range:

0000H÷00FFH -> selection SRAM-RTC (IC17)
0430H÷04FFH -> selection EEPROM 24c02 (IC16)
0430H÷05FFH -> selection EEPROM 24c04 (IC16)
0430H÷07FFH -> selection EEPROM 24c08 (IC16)

Procedure output description (values that must extract through POP ... instruction):

OUTPUT WRITING: 0 --> Writing OK.
 1 --> Wrong parameters.
 2 --> The device is not recognized.

OUTPUT READING: 0÷255 --> Reading OK.
 256 --> Wrong parameters.
 257 --> The device is not recognized.

Examples:

```
0   REM ***** Demo for WRITING the byte 85 at address 100 of serial EEPROM
10  BY_EE 1,100,85 : REM It selects WRITE operation
20  POP A : REM It reads the operation result
```

```
0   REM ***** Demo for READING the byte at address 100 of serial EEPROM
10  BY_EE 0,100,0 : REM It selects READ operation
20  POP A : REM It reads the value and the operation result
```

SERIAL EEPROM DATA BLOCK READ/WRITE PROCEDURE

Syntax: BL_EE <expr1>, <expr2>, <expr3>

Procedure description:

It performs a data block read or write operation at a specified address, on serial EEPROM (IC16) or on PCF 8583 (IC17). The W/R data buffer is located in EXTERNAL SRAM in **07E00H÷07EFF** addresses range.

Parameters description:

<expr1> --> 0 = Reading of a data block.
 --> 1 = Writing of a data block.
 <expr2> --> Initial location address (0÷<last device address>).
 <expr3> --> Number of bytes to write or read (1÷255).

Serial EEPROM and SRAM-RTC addresses range:

0000H÷00FFH	->	selection SRAM-RTC	(IC17)
0430H÷04FFH	->	selection EEPROM 24c02	(IC16)
0430H÷05FFH	->	selection EEPROM 24c04	(IC16)
0430H÷07FFH	->	selection EEPROM 24c08	(IC16)

Procedure output description (values that must extract through POP ... instruction):

OUTPUT:	0	-->	Reading/Writing OK.
	1	-->	Wrong parameters.
	2	-->	The device is not recognized.

Examples:

```
0   REM ***** Demo for WRITING 3 bytes starting from address 100 of SRAM RTC.
10  XBY(07E00H)=1
20  XBY(07E01H)=2
30  XBY(07E02H)=3
10  BL_EE 1,100,3 : REM Performs block write.
20  POP A : REM It reads the operation result.
```

```
0   REM ***** Demo for READING 3 bytes starting from address 100 of SRAM RTC.
10  XBY(07E00H)=0 : REM Sets reception buffer content to zero.
20  XBY(07E01H)=0
30  XBY(07E02H)=0
40  BL_EE 0,100,3 : REM Performs block read.
50  POP A : REM It reads the operation result.
60  FOR I=07E00H TO 07E02H : REM Prints Reception buffer content.
70  PRINT XBY(I),
80  NEXT I
```

SPECIAL FUNCTION REGISTERS READ OR WRITE PROCEDURE

Syntax: RW_SFR <expr1>, <expr2>, <expr3>

Procedure description:

It performs a special function register (SFR) read or write operations.

The User should remember that in "read procedure" the <expr3> parameter must be given even if it has no meaning. The SFR identification byte is a numeric code, with the following meaning:

SFR NAME	IDENTIFICATION SFR BYTE	
CTCON	0	
CTH3	1	
CTH2	2	
CTH1	3	
CTH0	4	
CMH2	5	
CMH1	6	
CMH0	7	
CTL3	8	
CTL2	9	
CTL1	10	
CTL0	11	
CML2	12	
CML1	13	
CML0	14	
IEN1	15	
IP1	16	
RTE	17	
S1ADR	18	
S1DAT	19	
S1STA	20	
S1CON	21	
STE	22	
TMH2	23	
TML2	24	
TM2CON	25	
TM2IR	26	
T3	27	
P4	28	
P5	29	(read only)

Parameters description:

<expr1> --> 0÷1 = R/W selection byte (0=Reading; 1=Writing).
 <expr2> --> 0÷29 = SFR identification byte.
 <expr3> --> 0÷255 = Byte to write (must be provided also for writing operations).

Procedure output description (values that must extract through POP ... instruction):

OUTPUT WRITING: 0 = Writing OK.
 65535 = Wrong parameters.

OUTPUT READING: 0÷255 = Read value.
 65535 = Wrong parameters.

Examples:

```
0  REM ***** Demo for writing byte 85 in P4
10 RW_SFR 1,28,85 : POP A
```

```
0  REM ***** Demo for P4 reading
10 RW_SFR 0,28,0 : POP A
20 IF A<>65535 THEN ? "P4= ",A
```

KEYBOARD MANAGEMENT PROCEDURE

Syntax: KEYB <expr>

Procedure description:

It enables or disables the matrix keyboard scanning and reads the possible key pressed code. This procedure can start or stop a periodic **QTP 24P** keyboard scanning, with debouncing on the pressed key, or it can return the pressed key code (0 if no key is pressed) through the stack. The keyboard managed by this procedure is a 4*6 matrix keyboard directly connected to connector **CN5** of **GPC® 553** I/O lines. The following table shows the ASCII code returned with relative I/O lines:

	CN5-PIN 11 (P1.4)	CN5-PIN 12 (P1.5)	CN5-PIN 9 (P1.6)	CN5-PIN 10 (P1.7)
CN5-PIN 5 (P4.5)	68	67	66	65
CN5-PIN 6 (P4.4)	72	71	70	69
CN5-PIN 3 (P4.3)	76	75	74	73
CN5-PIN 4 (P4.2)	54	52	51	50
CN5-PIN 1 (P4.1)	13	48	57	56
CN5-PIN 2 (P4.0)	55	27	53	49

Parameters description:

<expr> --> 0 = Keyboard scanning OFF.
 --> 1 = Keyboard scanning ON.
 --> 2 = Return the pressed key code (0 if no key is pressed) through the stack.
 The keyboard scanning is enabled if it was OFF.

Procedure output description (values that must extract through POP ... instruction):

OUTPUT: 0 = No key pressed.
 nn = Pressed key code (refer to the terminal keys map).
 65535 = Wrong parameters.

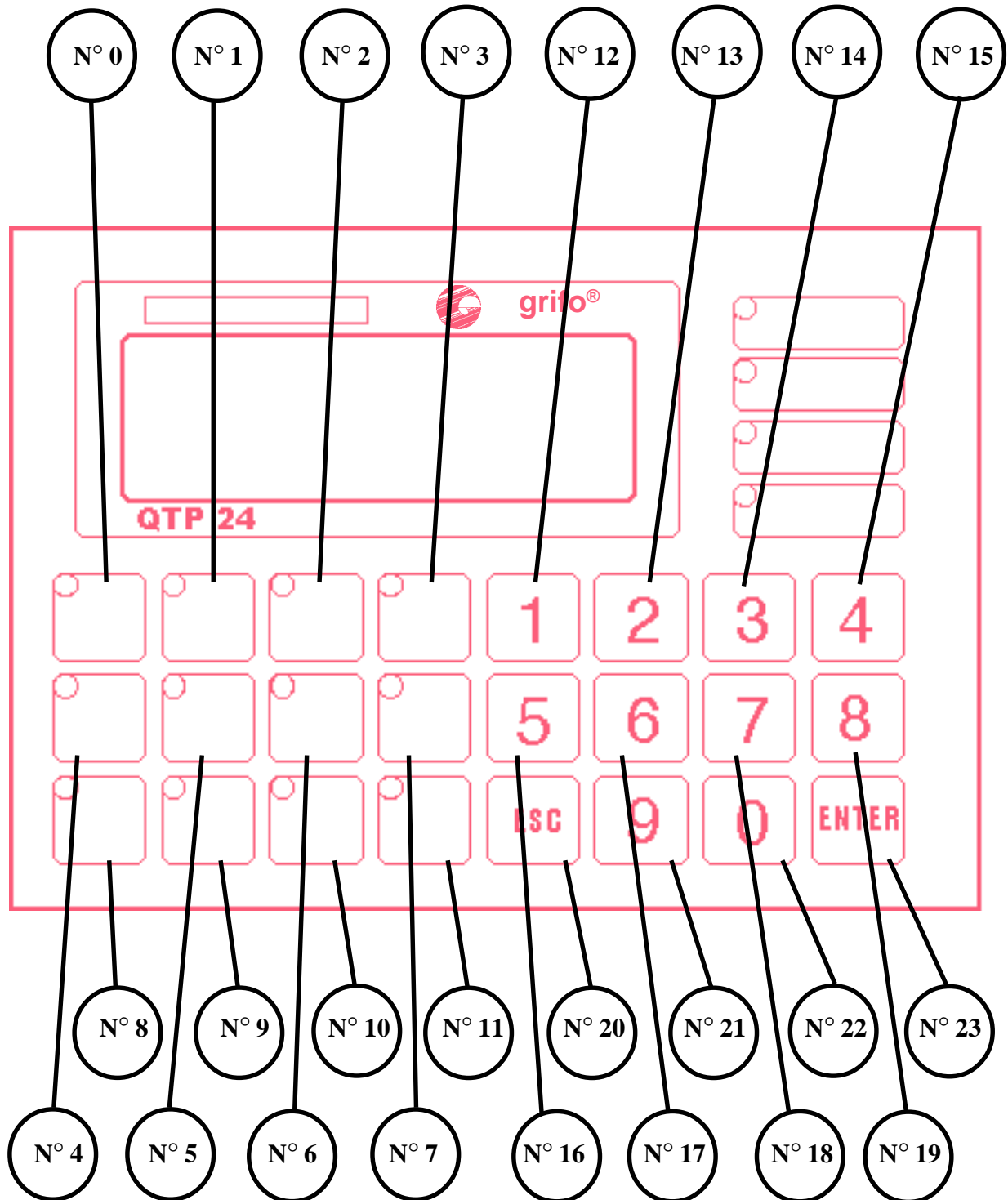
Example:

In the following demo program, it is enabled the matrix keyboard scanning and a loop is performed. In this loop the pressed key is controlled and when the 48 key code is pressed, the loop is terminated and the keyboard scanning is disabled.

```

0   REM ***** Matrix keyboard example
10  KEYB 1 : POP A : REM It enables keyboard scanning
20  KEYB 2 : POP A : REM It reads the key pressed code
30  IF A<>48 THEN 20
40  KEYB 0 : POP A : REM It disables keyboard scanning
50  END
    
```

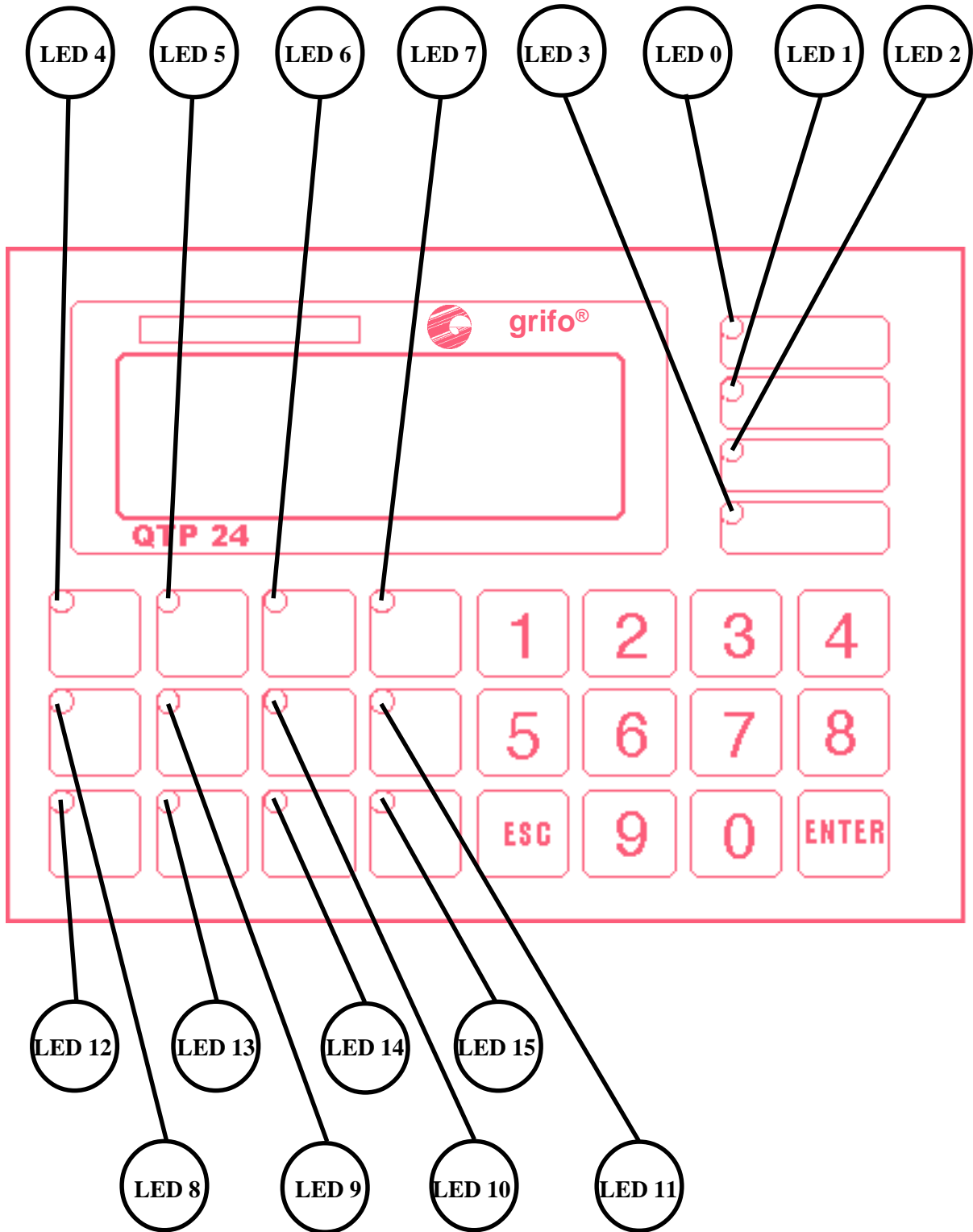
QTP 24P TERMINAL KEYS MAP



KEYS CODES ON QTP 24P

KEY N°	CODE	HEX CODE	MNEMONIC
0	65	41	A
1	66	42	B
2	67	43	C
3	68	44	D
4	69	45	E
5	70	46	F
6	71	47	G
7	72	48	H
8	73	49	I
9	74	4A	J
10	75	4B	K
11	76	4C	L
12	49	31	1
13	50	32	2
14	51	33	3
15	52	34	4
16	53	35	5
17	54	36	6
18	55	37	7
19	56	38	8
20	27	1B	ESC
21	57	39	9
22	48	30	0
23	13	0D	CR

QTP 24P LEDS MAP



DISPLAY SELECTION AND INITIALIZATION PROCEDURE

Syntax: DISPLAY <expr>

Procedure description:

Initializes the selected display. Please remark before using output redirection (UO 1) the User must invoke this function to select the display.

Parameters description:

<expr> --> 0 = FUTABA 20x2
1 = FUTABA 40x1
2 = FUTABA 40x2
3 = FUTABA 40x4
4 = LCD 20x2
5 = LCD 20x4
6 = LCD 40x1
7 = LCD 40x2
8 = LCD 40x4

Procedure output description (values that must extract through POP ... instruction):

OUTPUT: None.

Example:

```
10 DISPLAY 5 : REM Selects display LCD20x4.  
20 UO1 : REM OUTPUT redirection enabled  
30 PRINT "GRIFO", : REM Prints a string to the display  
40 UO0 : REM OUTPUT redirection disabled
```

SOFTWARE SERIAL LINE MANAGEMENT PROCEDURE

Syntax: COM2 <expr>

Procedure description:

This procedure manages all the operations on the software serial line (TXB=PIN 2 CN3B; RXB=PIN 5 CN3B). For the transmission on this line the user must use the PRINT@ ... instruction, while for the reception, there is a buffer (64 characters long) allocated in the EXTERNAL SRAM in 07F00H÷07F3FH addresses range. If the software serial line management is active, the User can't use the TIMER 0 instructions because this timer is used as baud rate generator.

Parameters description:

<expr> --> 0 = It disables the software serial line
 1 = It enables the software serial line at 1200 BAUD
 2 = It enables the software serial line at 2400 BAUD
 3 = It enables the software serial line at 4800 BAUD
 4 = It reads the number of characters already saved in the reception buffer
 5 = It resets the reception buffer

Procedure output description (values that must extract through POP ... instruction):

OUTPUT: 0 = Operation OK if <expr>=0, 1, 2, 3, 5.
 0÷63 = Number of received characters if <expr>=4.
 65535 = Wrong parameters.

Example:

```

0   REM ***** Demo for the software serial line
10  COM2 1 : POP B : REM It enables the software serial line management at 1200 BAUD
20  A=GET
30  IF A=0 THEN GOTO 70 : REM Has a character been received from the serial line ?
40  IF A=70 THEN GOTO 140 : REM Is "F" the received character ?
50  PRINT@ CHR(A), : REM Transmission of the character on the software serial line
60  GOTO 20
70  COM2 4 : POP A : REM Request of the number of received characters
80  IF A<>1 THEN 20
90  FOR I=0 TO A-1
100 PRINT CHR(XBY(07F00H+I)),
110 NEXT I
120 COM2 5 : POP B : REM It resets the reception buffer
130 GOTO 20
140 COM2 0 : POP B : REM It disables the software serial line management
150 END
  
```

RTC INTERRUPT MANAGEMENT PROCEDURE

Syntax: ALARM <expr1>, <expr2>, <expr3>, <expr4>, <expr5>, <expr6>, <expr7>, <expr8>

Procedure description:

This procedure enables the RTC (IC17) interrupt, so it is possible to use this procedure to generate time bases or manage alarms. For further informations about how to use the RTC, please refer to the manufacturer documentation.

Please remark that RTC interrupt management is possible on **GPC® 553** only connecting jumper J13 in position 1-2. In this situation the RTC interrupt signal will be connected to CPU signal /INT1, which will be managed by function **ONEX1** of **BASIC 553**.

Parameters description:

<expr1> --> 0 = Sets NO CLOCK ALARM (ALARM MODE).
 1 = Sets DAILY ALARM (ALARM MODE).
 2 = Sets WEEKDAY ALARM (ALARM MODE).
 3 = Sets DATED ALARM (ALARM MODE).
 4 = Sets TIMER (TIMER MODE).
 5 = Resets ALARM flag.

TIMER MODE

<expr2> --> 0÷99 = Count value.
 <expr3> --> 0 = No timer.
 1 = Counts "HUNDREDTH OF SECOND".
 2 = Counts "SECONDS".
 3 = Counts "MINUTES".
 2 = Counts "HOURS".
 2 = Counts "DAYS".

ALARM MODE

<expr2> --> Value for hours (0÷23).
 <expr3> --> Value for minutes (0÷59).
 <expr4> --> Value for seconds (0÷59).
 <expr5> --> Value for day of week (0÷6).
 <expr6> --> Value for day of month (1÷31).
 <expr7> --> Value for month (1÷12).
 <expr8> --> Value for year (0÷3).

Procedure output description (values that must extract through POP ... instruction):

OUTPUT: 0 = OK
 1 = Error accessing RTC.
 2 = Wrong parameters.

Examples:

```
0   REM ***** Demo for generating a time base of 1 second.
10  MTOP=07DFFH
20  B=1 : REM Initialization interrupts number counter.
30  ALARM 4, 1, 2, 0, 0, 0, 0, 0 : POP C : REM Trigger on interrupt each second.
40  ONEX1 100
50  GOTO 50
100 ? "Number of interrupts counted= ", B
110 B=B+1
120 ALARM 5, 0, 0, 0, 0, 0, 0, 0 : POP C : REM Resets alarm flag
130 RETI

0   REM ***** Demo for generating an interrupt at a specified date and time.
10  MTOP=07DFFH
20  ALARM 3, 12, 30, 0, 3, 25, 12, 1 : POP C : REM Interrupt on December 25th 1997 12:30:00.
30  ONEX1 100
40  GOTO 40
100 ? "Alarm triggered."
110 ALARM 5, 0, 0, 0, 0, 0, 0, 0 : POP C : REM Resets alarm flag
120 RETI
```

USER OUTPUT REDIRECTION PROCEDURE

BASIC 553 includes specific software procedures to solve the User interface problems. In the standard industrial applications, the User interface is realized with displays and custom keyboards; so **BASIC 553** provides high level procedure for these components. Normally BASIC performs User interface or console operations through the target card serial line, but at the same time it has the possibility to redirect console operations to other hardware devices, thanks to specific software (for further information, please read UO0, UO1 instructions).

The User output can be redirected to fluorescent or LCD displays, simply using the PRINT instruction.

Example:

```
10  DISPLAY 5 : REM LCD20x4display selection.
20  UO1 : REM It enables output redirection.
30  PRINT "Hello", : REM It prints a string on the display.
40  PRINT "Grifo", : REM It prints a string on the display.
50  PRINT CHR(1), : REM Cursor in HOME position.
60  PRINT CHR(27), : PRINT CHR(80), : REM Cursor OFF.
70  UO0 : REM It disables output redirection.
```

QTP 24P shows on its display all the characters having a code included in the range **32÷255 (20÷FF Hex)**; if it is sent a code not included in this range and this latter is not a command, it is ignored.

The characters of the codes in the range **32÷127 (20÷7F Hex)** correspond to the ones of the standard ASCII table, while characters associated to **128÷255 (80÷FF Hex)** codes, may show different symbols depending on the type of the display installed. This is the reason why it is opportune to refer to the Tables at the end of this documentation.

The character is visualized in the at-the-moment cursor position and this latter will go to the next position; if it is placed in the last character down on the right of the display, it will be placed to the Home position.

COMMANDS FOR CURSOR POSITIONING

CURSOR LEFT

Code: 21 (15Hex)
Mnemonic: NACK

The cursor is shifted of one position on the left without modifying the display contents. If the cursor is in Home position, it will be placed in the last character at the down-right position of the display.

CURSOR RIGHT

Code: 06
Mnemonic: ACK

The cursor is shifted of one position on the right.
If the cursor is placed on the last display character, down-right, it will be placed on the Home position.

CURSOR DOWN

Code: 10 (0A Hex)
Mnemonic: LF

The cursor will be placed on the next line of that one it is now but it will remain in the same column.
If the cursor is in the last display line, it will be placed at the first display line.

CURSOR UP

Code: 26 (1A Hex)
Mnemonic: SUB

The cursor will be placed in the previous line of that one it is now, but it will remain in the same column. If the cursor is on the first display line, it will be placed on the last display line.

HOME

Code: 01
Mnemonic: SOH

The cursor is on Home position i.e first line, first column of the display.

CARRIAGE RETURN

Code: **13** **(0D Hex)**
Mnemonic: **CR**

The cursor is placed at the beginning of the line where it is.

CARRIAGE RETURN+LINE FEED

Code: **29** **(1D Hex)**
Mnemonic: **GS**

The cursor is placed at the beginning of next line at which it was placed.
If the cursor is at the last display line, it will be placed at the beginning of the first line i.e Home position.

CURSOR ABSOLUTE POSITIONING WITH 20H OFFSET

Code: **27 89 r c** **(1B 59 r c Hex)**
Mnemonic: **ESC Y ASCII(r) ASCII(c)**

The cursor is placed at the absolute point indicated through "r" and "c".
These codes are referred to line and column values of the display at which the **32 (20 Hex)** offset must be add. If, for example, you wish to place the cursor at Home position (0 line, 0 column) the next sequence is necessary **27 89 32 32**.
If line and column values are not compatible to the installed display, that command is ignored.

COMMANDS FOR CHARACTERS ERASURE

BACKSPACE

Code: **08**
Mnemonic: **BS**

The cursor shifts a character on the left by erasing the contents of the reached cell. If the cursor is at Home position, the character placed in the last cell, down- on the right of the display, will be erased.

CLEAR PAGE

Code: **12** **(0C Hex)**
Mnemonic: **FF**

Complete erasure of the display and the cursor returns to Home position.

CLEAR LINE

Code: **25** **(19 Hex)**
Mnemonic: **EM**

The complete line where cursor is placed is erased and then the cursor goes at the beginning of the said line.

CLEAR END OF LINE

Code: **27 75** **(1B 4B Hex)**
Mnemonic: **ESC K**

All characters on the line where the cursor is placed are erased starting from cursor position up to the end of the line.

The cursor stays on the position as it was when **Clear End of Line** code arrives.

If, for example, the cursor is at the beginning of a display line, the complete line will be erased.

CLEAR END OF PAGE

Code: **27 107** **(1B 6B Hex)**
Mnemonic: **ESC k**

All characters starting from the Cursor point up to the end of the display, are erased. The cursor stays in the same position as it was before the **Clear end of Page** code arrival. If, for example, the cursor is at Home position, the display will be completely erased.

COMMANDS FOR CURSOR ATTRIBUTES MANAGEMENT

CURSOR OFF

Code: 27 80 (1B 50Hex)
Mnemonic: ESC P

The cursor is not active and it is not more visible.

STATIC CURSOR ON

Code: 27 79 (1B 4F Hex)
Mnemonic: ESC O

The cursor is started so it is visible. Now it is a not blinking line placed under the char.

Note: this command is not available if **Futaba 40x4** display is installed.

BLINKING "UNDERLINE" CURSOR

Code: 27 77 (1B 4D Hex)
Mnemonic: ESC M

The cursor is started so it is visible. Now it is a blinking line placed under the char.

Note: This command is available only for **Futaba** displays : **20x2** and **40x1** type.

BLINKING "BLOCK" CURSOR

Code: 27 81 (1B 51 Hex)
Mnemonic: ESC Q

The cursor is started so it is visible. Now it is a blinking rectangular form and it is alternatively visualized with the char put on the same.

Note: This command is available only for **LCD** displays otherwise is ignored

COMMANDS FOR LEDS MANAGEMENT

LED ACTIVATION

Code: 27 50 n.LED Attr. (1B 32 n.LED Attr. Hex)
Mnemonic: ESC 2 ASCII(n.LED) ASCII(Attr.)

The LED shown in "n.LED" with the specified attribute in "Attr." is started.

If you use the **QTP 24P** card LEDs numbers are included in a range of **0÷15** as shown in the picture of the card.

The attributes available are as follows:

0	<i>Not enabled LED</i>
255 (<i>FF Hex</i>)	<i>Enabled LED</i>
85 (<i>55Hex</i>)	<i>Blinking LED</i>

ex. If you wish to enable LED n.5 with blinking attribute, the following sequence has to be sent: **27 50 5 85**.

If the parameter with LED number or that one with the attribute, it is not valid, the command is ignored.

LEDS MASK ACTIVATION

Code: 27 52 byte1 byte2 byte3 (1B 34 byte1 byte2 byte3 Hex)
Mnemonic: ESC 4 ASCII(byte1) ASCII(byte2) ASCII(byte3)

All **QTP 24P** LEDs are contemporarily managed as indicated in "byte1", "byte2" and "byte3" following this code:

byte1 (bit 0...7)	LED 0...LED 7
byte2 (bit 0...7)	LED 8...LED 15
byte3 (bit0...5)	Reserved

If a bit is placed in 0 position, the correspondent LED is OFF, viceversa it will be ON if the correspondent bit is on 1 position.

If there are some LEDs having the blinking attribute, this latter will be disabled.

For **QTP 24P** the "byte3" must be always sent even if it has no meaning for running the 16 LEDs of the said terminal.

COMMANDS FOR KEYBOARD MANAGEMENT

KEY RECONFIGURATION

Code: 27 55 key no. code (1B 37 key no. code Hex)
Mnemonic: ESC 7 ASCII(key no.) ASCII(code)

When the selected key is reconfigured, each time it is pressed, the card will send the new specified code in serial mode. The number of the key to be reconfigured is obtainable by looking at the **QTP 24P** picture and it must be included in a range of **0÷23 (0÷17 Hex)** if this is not done the command is ignored. The code value can vary in a range of **0÷254 (0÷FE Hex)** as the **255** value (**FF Hex**) indicates that the key must be disabled so when it is pressed the card will not send any codes.

P.S.

The said command needs a data writing on the on-board EEPROM so before executing it, it is better to be sure that the card is ready for a new writing on such device otherwise the command will be ignored.

BXC51 COMPILER REL. 5.0 USE

BXC51 compiler Rel. 5.0 is a software tools used to compile a **BASIC 52** source program obtaining a directly executable code for the target card. In this way the application program execution speed is notably incremented, in fact the program is not interpreted by **BASIC 52**, but directly executed by microprocessor.

BXC51 can be used with the **BASIC 553** new features, typing some options on the command line as described below:

BXC51 -2 -b<BAUD> -c7F00 -u7DFF -bCRUNxxyy <File Name> (SRAM=32K)

CRUNxxyy.BXL is a library delivered by **grifo®** where xx indicates the crystal frequency and yy is the version number. This library is supplied with BXC51 package.

The option -b<BAUD> inform compiler that automatic baud rate search routine must not be inserted in the generated code and it must initialize serial line with a fixed baud rate = BAUD. The allowed value for BAUD parameter are: 1200, 2400, 4800, 9600, 19200.

Please remark that if the crystal frequency is 22118400 Hz the real baud rate will be twice the declared baud rate. For example: if the declared baud rate is 19200 (-b19200) then the real baud rate will be 38400 BAUD.

If the frequency of the on board crystal is not multiple of 11059200 Hz, the option -b<BAUD> has the only effect to avoid to generate the code for automatic baud rate detection.

Moreover to set correctly the desired baud rate, in the BASIC source program must be added the instruction to set directly the internal microprocessor register. Here follows an example for a 14.7456 MHz card:

```
0   REM ***** Original source program
10  PRINT "Hello word"

0   REM ***** Source program modified for BXC51 compiler
5   TIMER1=0FCFCH : REM Baud=19200.
10  PRINT "Hello word"
```

For further information on the other **BXC51** command line options please refer to BXC51 documentation.

QTP 24P COMMAND CODES SUMMARY TABLE

COMMAND	CODE	HEX CODE	MNEMONIC
HOME	01	01	SOH
CURSOR LEFT	21	15	NACK
CURSOR RIGHT	06	06	ACK
CURSOR DOWN	10	0A	LF
CURSOR UP	26	1A	SUB
CARRIAGE RETURN	13	0D	CR
CR+LF	29	1D	GS
Cursor absolute positioning with 20H OFFSET	27 89 r c	1B 59 r c	ESC Y ASCII(r) ASCII(c)
BACKSPACE	08	08	BS
CLEAR PAGE	12	0C	FF
CLEAR LINE	25	19	EM
CLEAR END OF LINE	27 75	1B 4B	ESC K
CLEAR END OF PAGE	27 107	1B 6B	ESC k
Cursor OFF	27 80	1B 50	ESC P
Static cursor ON	27 79	1B 4F	ESC O
Blinking "UNDERLINE" cursor	27 77	1B 4D	ESC M
Blinking "BLOCK" cursor	27 81	1B 51	ESC Q
LED activation	27 50 n.LED Attr.	1B 32 n.LED Attr.	ESC 2 ASCII(n.LED) ASCII(Attr.)
LEDS mask activation	27 52 byte1 byte2 byte3	1B 34 byte1 byte2 byte3	ESC 4 ASCII(byte1) ASCII(byte2) ASCII(byte3)
Key reconfiguration	27 55 key no. cod.	1B 37 key no. cod.	ESC 7 ASCII(key no.) ASCII(cod.)

DISPLAYS CHARACTERS TABLES

		Higher 4-bit (D4 to D7) of Character Code (Hexadecimal)																
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
Lower 4-bit (D0 to D3) of Character Code (Hexadecimal)	0	CG RAM (1)	!	"	#	\$	%	&	'	()	*	+	,	-	.	:	
	1	CG RAM (2)	;	<	=	>	?	@	A	B	C	D	E	F	G	H	I	J
	2	CG RAM (3)	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
	3	CG RAM (4)	[\]	^	_	`	{		}	~						
	4	CG RAM (5)																
	5	CG RAM (6)																
	6	CG RAM (7)																
	7	CG RAM (8)																
	8	CG RAM (1)																
	9	CG RAM (2)																
	A	CG RAM (3)																
	B	CG RAM (4)																
	C	CG RAM (5)																
	D	CG RAM (6)																
	E	CG RAM (7)																
	F	CG RAM (8)																

LCD 20x2 CHARACTERS TABLE



MSB	0000	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
XXXX0000	LSB E5 E4M E3D		0	1	2	3	4			5	6	7	8	9	A
XXXX0001	0	!	"	#	\$	%	&			'	()	*	+	,
XXXX0010	1	.	:	;	<	=	>			?	@	A	C	E	G
XXXX0011	2	I	O	S	C					J	T	F	E		
XXXX0100	3	4	O	T	A					V	X	Y	Z	[]
XXXX0101	4	^	_	0	1	2	3			4	5	6	7	8	9
XXXX0110	5	:	;F	V	V					8	9	C	B		
XXXX0111	6	'	7	G	H	I	J			K	L	M	N	O	P
XXXX1000	7	Q	R	X	X	X				4	5	6	7	X	X
XXXX1001	8)	9	V	V					8	9	A	B		
XXXX1010	9	*	#	J	Z	Z				8	9	X	Y		
XXXX1011	0	+	X	X	X	X				4	5	6	7	X	X
XXXX1100	1	.	<	L	R	I				8	9	0	1	2	3
XXXX1101	2	-	=	M	N	O				4	5	6	7	X	X
XXXX1110	3	=	>	X	X	X				8	9	0	1	2	3
XXXX1111	4	/	?	0	1	2				8	9	0	1	2	3

LCD 20x4 AND 40x2 CHARACTERS TABLE



LOWER 4-BIT HEXADECIMAL

Upper 4 bits Lower 4 bits	0000 (0)	0010 (2)	0011 (3)	0100 (4)	0101 (5)	0110 (6)	0111 (7)	1010 (A)	1011 (B)	1100 (C)	1101 (D)	1110 (E)	1111 (F)
xxxx0000 (0)	CG RAM (1)		0	A	P	'	F	-	9	E	o	p	
xxxx0001 (1)	(2)	!	1	A	Q	a	q	#	T	T	4	ä	q
xxxx0010 (2)	(3)	"	2	B	R	b	r	7	U	X	P	B	
xxxx0011 (3)	(4)	#	3	C	S	c	s	u	o	T	E	e	e
xxxx0100 (4)	(5)	*	4	D	T	d	t	.	I	I	T	u	a
xxxx0101 (5)	(6)	%	5	E	U	e	u	=	o	*	u	e	o
xxxx0110 (6)	(7)	@	6	F	V	f	v	o	o	o	o	P	E
xxxx0111 (7)	(8)	"	7	G	W	g	w	7	+	X	o	g	o
xxxx1000 (8)	(1)	<	8	H	X	h	x	4	o	*	u	o	X
xxxx1001 (9)	(2)	>	9	I	Y	i	y	o	o	u	u	o	u
xxxx1010 (A)	(3)	*	:	J	Z	j	z	o	o	o	o	u	o
xxxx1011 (B)	(4)	+	:	K	C	k	c	o	o	E	o	o	o
xxxx1100 (C)	(5)	.	<	L	*	l	l	o	o	o	o	o	o
xxxx1101 (D)	(6)	-	=	M	I	m	i	u	o	o	o	o	o
xxxx1110 (E)	(7)	.	>	N	^	n	+	a	e	o	o	o	o
xxxx1111 (F)	(8)	/	?	O	_	o	+	o	o	o	o	o	o

LCD 40x1 AND 40x4 CHARACTERS TABLE



	D7	D6	D5	D4	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	0	0	0	0	DP															
0001	1																			
0010	2																			
0011	3																			
0100	4	DIM	CU1																	
0101	5		CU2																	
0110	6		CU3																	
0111	7		DC																	
1000	8	BS																		
1001	9	HT																		
1010	A																			
1011	B																			
1100	C																			
1101	D	CLR																		
1110	E																			
1111	F	ALD	RST																	

FLUORESCENT 20x2 CHARACTERS TABLE



	D 7	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
	D 6	0	0	0	0	1	1	1	1	0	0	0	1	1	1	1	1
	D 5	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
	D 4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
3210 DDDD		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	0	!	!	0	0	P	^	P	A	E		-	9	E	W	P	
0001	1	!	!	1	A	Q	a	4	A	w	.	7	7	Q	A	9	
0010	2	!"	"	2	B	R	b	r	A	E	T	4	W	X	E	0	
0011	3	!"#	#	3	C	S	c	s	A	R	U	9	T	E	e	w	
0100	4	!"#\$	\$	4	D	T	d	t	A	@	.	I	T	P	M	Q	
0101	5	!"#\$%	%	5	E	U	e	u	E	O	.	+	+	I	e	U	
0110	6	!"#\$%&	&	6	F	V	f	v	O	+	7	+	+	+	+	+	
0111	7	!"#\$%&'	'	7	G	W	g	w	O	+	7	+	+	+	+	+	
1000	8	!"#\$%&'((8	H	X	h	x	O	+	7	+	+	+	+	+	
1001	9	!"#\$%&'())	9	I	Y	i	y	O	+	7	+	+	+	+	+	
1010	A	!"#\$%&'() *	*	A	J	Z	j	z	O	+	7	+	+	+	+	+	
1011	B	!"#\$%&'() * +	+	B	K	Z	k	z	O	+	7	+	+	+	+	+	
1100	C	!"#\$%&'() * + ,	,	C	L	#	l	#	O	+	7	+	+	+	+	+	
1101	D	!"#\$%&'() * + , -	-	D	M	^	m	^	O	+	7	+	+	+	+	+	
1110	E	!"#\$%&'() * + , - .	.	E	N	^	n	^	O	+	7	+	+	+	+	+	
1111	F	!"#\$%&'() * + , - . /	/	F	O	_	o	_	O	+	7	+	+	+	+	+	

FLUORESCENT 20x4 CHARACTERS TABLE



	D7	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
	D6	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
	D5	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
	D4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
03020100		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0 0 0 0	0	DP		0	a	P	'	p	o	e	h	n	e	a	a	z	
0 0 0 1	1	DC1	!	1	A	a	g	o	h	f	a	e	e
0 0 1 0	2	DC2	"	2	B	R	B	r	a	a	a	r	r	r	r	r	#
0 0 1 1	3	DEF	#	3	C	S	c	s	a	c	c	l	e	y	l		
0 1 0 0	4	DIM	\$	4	D	T	a	t	e	u	n	e	x	t	#	*	
0 1 0 1	5		%	5	E	e	e	e	a	a	a	r	r	r	r	r	*
0 1 1 0	6		&	6	F	v	v	v	a	a	a	e	e	e	e	e	e
0 1 1 1	7		'	7	G	w	w	w	c	c	c	x	e	t	w	e	
1 0 0 0	8	BS	(8	H	x	x	x	e	y	c	r	x	b	w	e	
1 0 0 1	9	HT)	9	I	v	v	v	e	d	r	t	t	t	t	t	+
1 0 1 0	A	LF	*	A	J	z	z	z	a	d	r	p	t	b	a	+	
1 0 1 1	B		+	B	K	e	e	e	e	e	e	e	e	e	e	e	*
1 1 0 0	C		,	C	L	l	l	l	e	e	e	r		a	a	a	*
1 1 0 1	D	CR	-	D	=	m	m	m	n	n	n	i	*	UF0	a	a	
1 1 1 0	E		.	E	>	n	n	n	r	r	r	e	o	UF1	e		
1 1 1 1	F	RST	/	F	?	o	o	o	o	o	o	o	o	UF2	e	e	

FLUORESCENT 40x1 AND 40x2 CHARACTERS TABLE

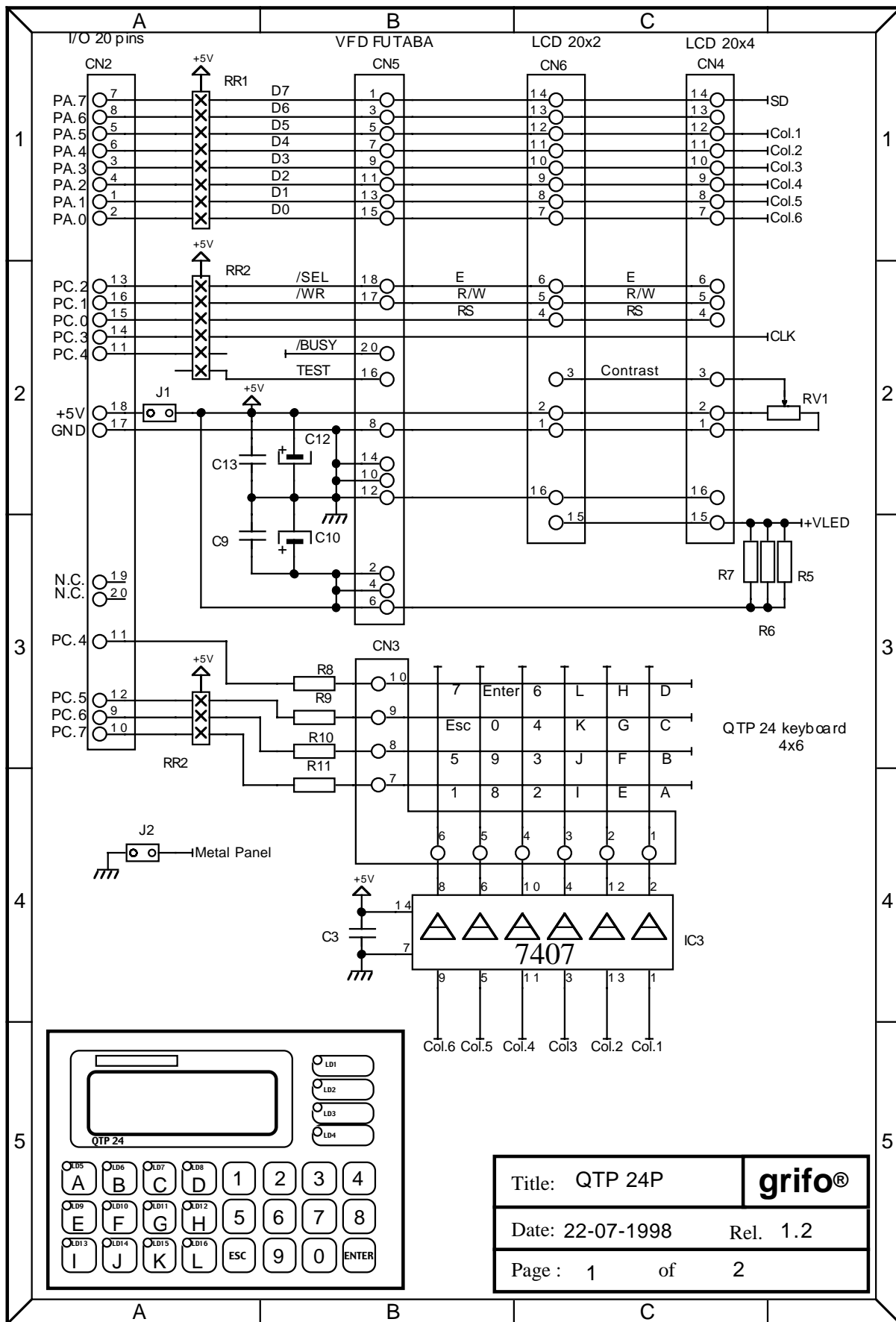


	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000		DP		0	1	2	3	4	5	6	7	8	9	:	;	<
xxxx0001		DC1	!	1	2	3	4	5	6	7	8	9	:	;	<	>
xxxx0010		DC2	"	2	3	4	5	6	7	8	9	:	;	<	>	~
xxxx0011	DEF	DC3	#	3	4	5	6	7	8	9	:	;	<	>	~	^
xxxx0100	DIM	DC4	\$	4	5	6	7	8	9	:	;	<	>	~	^	^
xxxx0101		DC5	%	5	6	7	8	9	:	;	<	>	~	^	^	^
xxxx0110			0	1	2	3	4	5	6	7	8	9	:	;	<	>
xxxx0111			7	8	9	:	;	<	>	~	^	^	^	^	^	^
xxxx1000	BS		0	1	2	3	4	5	6	7	8	9	:	;	<	>
xxxx1001			1	2	3	4	5	6	7	8	9	:	;	<	>	~
xxxx1010		UP	*	2	3	4	5	6	7	8	9	:	;	<	>	~
xxxx1011	HM CLR	DWN	+	3	4	5	6	7	8	9	:	;	<	>	~	~
xxxx1100		RT	.	<	3	4	5	6	7	8	9	:	;	<	>	UF0
xxxx1101	CR + LF	LT	—	3	4	5	6	7	8	9	:	;	<	>	UF1	UF1
xxxx1110			.	>	3	4	5	6	7	8	9	:	;	<	>	UF2
xxxx1111		RST	/	?	0	1	2	3	4	5	6	7	8	9	:	<

FLUORESCENT 40x4 CHARACTERS TABLE

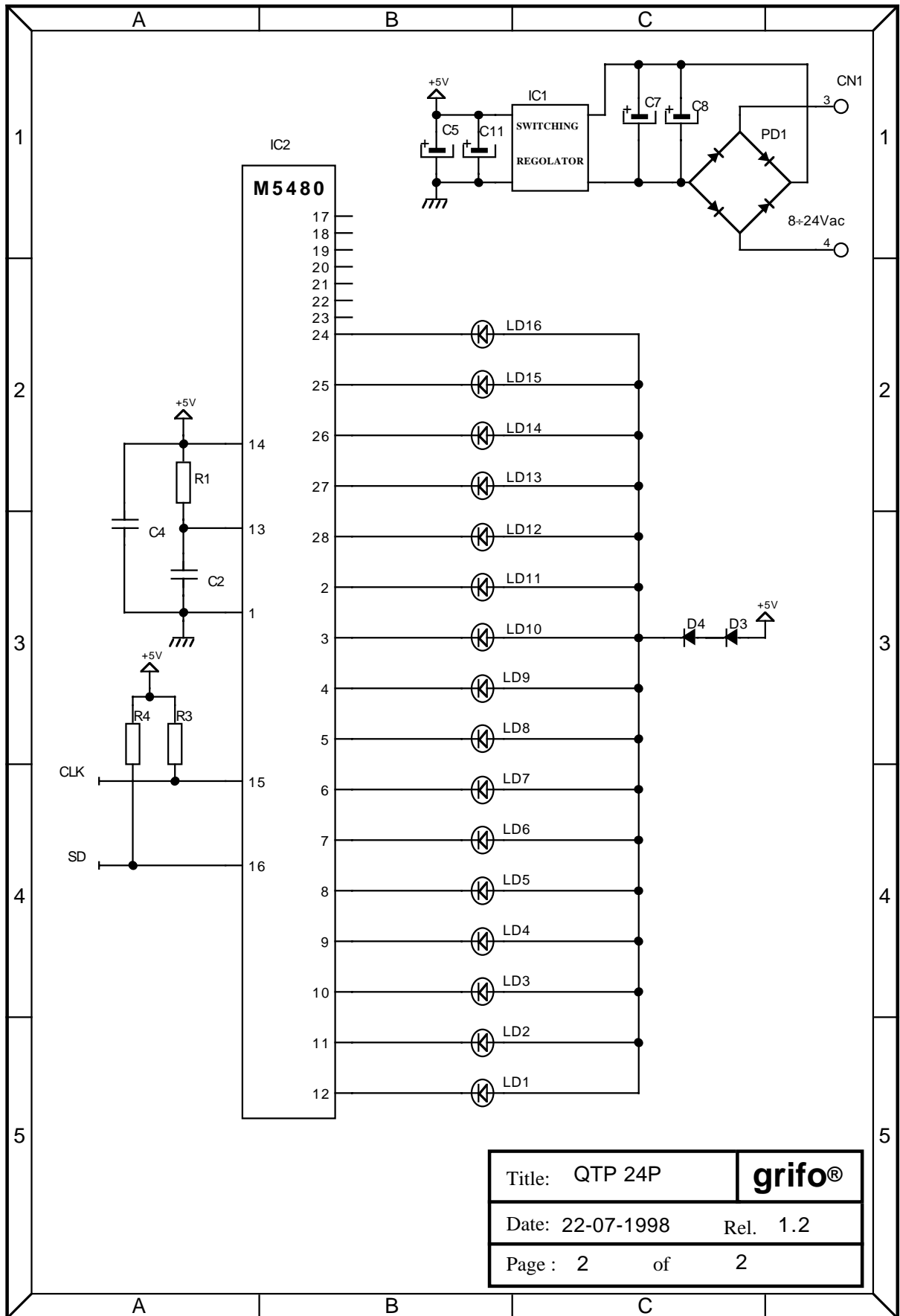


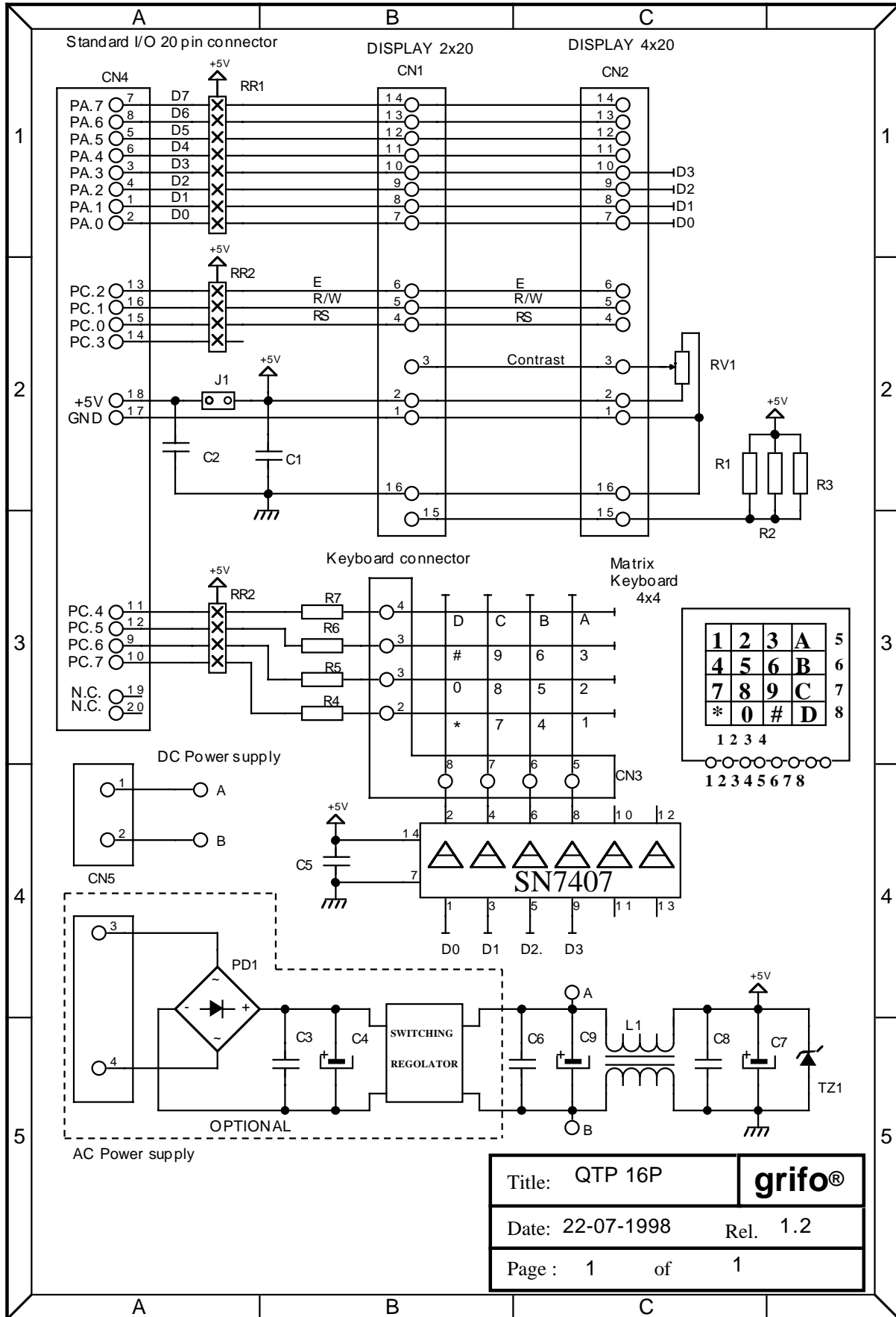
ELECTRIC DIAGRAM



Title: QTP 24P	grifo®
Date: 22-07-1998	Rel. 1.2
Page : 1	of 2

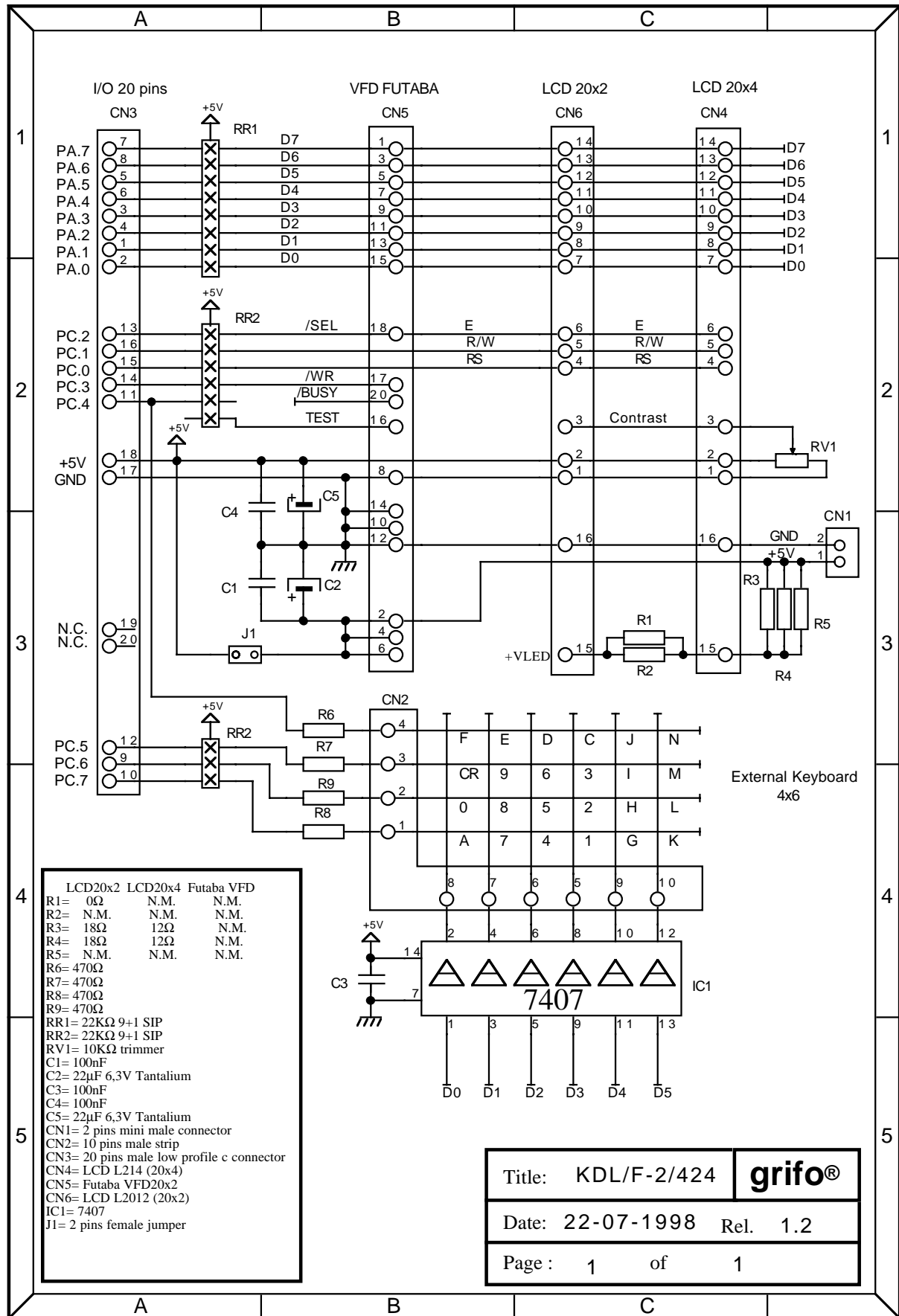






Title: QTP 16P	grifo®
Date: 22-07-1998	Rel. 1.2
Page : 1	of 1





Title: KDL/F-2/424	grifo®
Date: 22-07-1998	Rel. 1.2
Page: 1	of 1



