

Keypad Display Wildcard

The Keypad/Display Wildcard™ provides a convenient interface to a 4 x 20 character display and 4 x 5 keypad. Combined with Mosaic's QCard, QScreen or QVGA Controllers, it is an ideal solution for hand-held or space-constrained applications that require a programmable embedded computer and a low-cost yet smart user interface.

- ⇒ *Measuring only 2" x 2.5", the Keypad/Display Wildcard mounts directly on a QCard Controller.*
- ⇒ *This Wildcard is shipped with a 4x20 LED-backlit character display and 4x5 keypad, plus a simple ribbon cable interface for custom placement of the keypad and display in your instrument.*
- ⇒ *It has an additional field header that brings out 4 nibble-wise programmable input/output lines and 4 input lines.*
- ⇒ *This WildCard also contains an onboard 2 KHz buzzer to provide audible feedback for keypad presses, or any other purpose.*

The Keypad/Display WildCard provides a hardware and software interface for a 5x4 keypad and 4x20 liquid crystal display (LCD). These devices connect to the WildCard via a simple "straight-through" ribbon cable interface. Pre-coded routines in the QED-Forth kernel (available to both C and Forth programmers) scan the keypad and write to the LCD display.

Table 1-1 Technical Specifications

Property	Value
Power	5 VDC derived from the WildCard bus 0.5 W using the non-backlit display), or, 2 W using an LED-backlit display.
Keypad	5 column by 4 row keypad, tactile feedback, snap-on domes for user-configurable legends, mounting hole size 2.7" x 3.0", standard Grayhill part, interchangeable with other sizes.
Display	4 line by 20 character LCD display with optional LED backlight
Backlight	LED backlight with software ON/OFF.
Beeper	Software controlled 2 KHz, 0.2 W buzzer at 80 dB
General Purpose I/O	4 input lines and 4 lines programmable together as all inputs or all outputs
Output current capability:	4mA source, 24mA sink

Property	Value
Connectors	34-pin dual row 0.1" pitch keypad/display connector 24-pin dual row 0.1" pitch field I/O connector

Connecting the WildCard

The WildCard is shown in Figure 1.1. On the right the WildCard Port Header, H1, connects to the QCard or QScreen Controller, and on the left the Field Header, H4, provides 8 lines of digital I/O. The 34-pin connector on the side, H2, connects to a ribbon cable to the keypad and display.

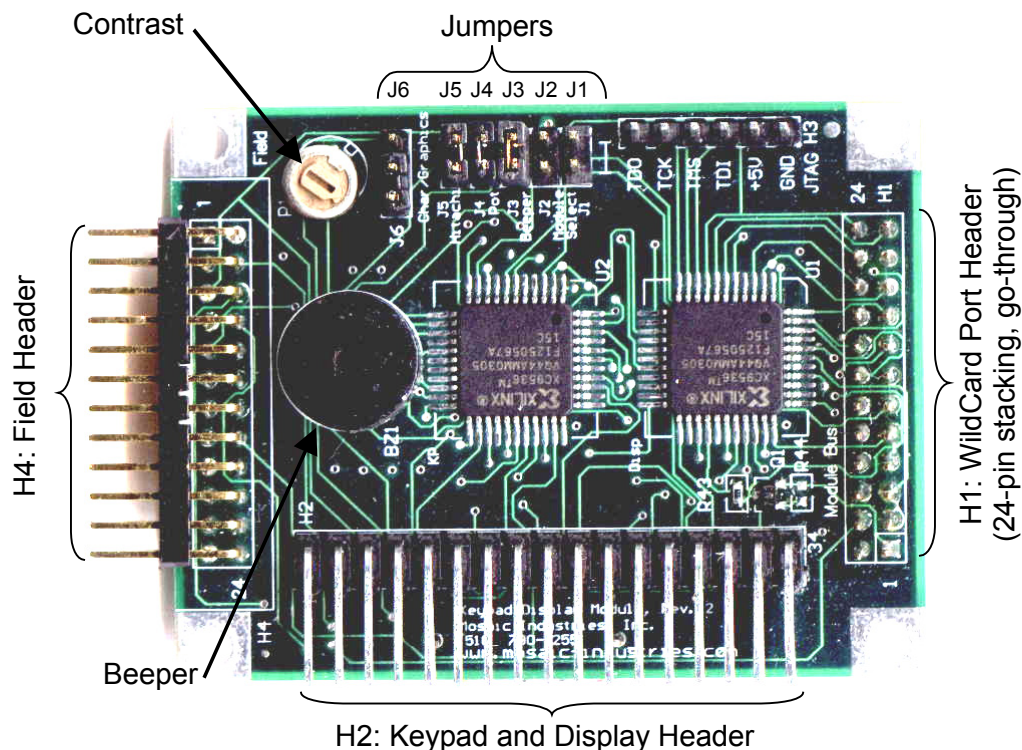


Figure 1-1 Headers and Jumpers on the Keypad/Display WildCard.

Mounting on a QCard, QScreen, or QVGA Controller

With the power off, the WildCard may be mounted on a QCard, QScreen, QVGA Controller, or PowerDock by directly plugging connector H1 into a WildCard Port connector on the controller. The corner mounting holes on the module should line up with the standoffs on the controller.

Setting the Module Address

Each WildCard Port on a QCard, QScreen or QVGA Controller accommodates up to four WildCards, at four different module addresses: 0, 1, 2, or 3 if installed on Port 0, and 4, 5, 6, or 7 if installed on Port 1. A QCard has only one module port, Port 0. Even so, if the QCard is mounted

on a PowerDock it can hold four WildCards on the QCard, and four more on Port 1 of the PowerDock. You should set the module addresses on the WildCards in your system so that they do not conflict, that is, so that no two modules on the same port have the same address (jumper settings). Jumpers J1 and J2 allow you to select the module address.

In order to use the built-in driver routines the Keypad/Display WildCard must be configured at module address 0. In some applications you may wish to forgo use of the built-in drivers and configure the WildCard at another module address. In that case, use Table 1-2 to determine the jumper settings you need for each module address. But to make use of the internal driver routines, you must configure the WildCard for module address 0. Make sure that there are no jumpers set on J1 and J2.

Table 1-2 Jumper settings and addresses.

Module Port	Module Address	Page Address	Installed Jumper Shunts
0	0	0x00	None
	1	0x01	J1
	2	0x02	J2
	3	0x03	J1 and J2
1	4	0x04	None
	5	0x05	J1
	6	0x06	J2
	7	0x07	J1 and J2

Setting the Jumpers

There are six jumpers on the board allowing you to select various options. Table xx describes their purposes and default settings.

Table 1-3 Default jumper positions.

Jumper	Default Setting	Meaning
J1	Removed	Module address bit. Together with J2 chooses module addresses of 0-3. Should be removed in normal operation to choose page 0 addresses.
J2	Removed	Module address bit. Should be removed in normal operation to choose page 0 addresses.
J3	Connected	Enables the onboard beeper. Remove to disable the beeper.
J4	Connected	Enables the onboard contrast adjustment potentiometer. Remove if an external pot is used to adjust the display contrast.
J5	Connected	Chooses the display type: connected for an Hitachi character or graphics display controller; or removed for a Toshiba graphics display controller. Standard WildCards ship with an Hitachi-type character display.

Jumper	Default Setting	Meaning
J6	Set to "Char"	Chooses either a 4x20 character or a graphics display. Standard WildCards ship with a character display.

Connecting a Keypad and Display

The keypad/display interface connector, H2 in the photo, is a 34-pin dual-row right-angled header. A 34-line ribbon cable plugs into this header. At the other end of the cable are two female single-row connectors: one at the end for the keypad, and the other in the middle for the display.

The 34-pin female connector should be plugged into H2 with the ribbon exiting the connector toward the board. Line 1 in the ribbon cable (on the edge of the cable that the 10-pin keypad connector is aligned along) mates with pin 1 of H2, closest to the field header, H4.

Connecting the Keypad

The 10-pin connector on the end of the ribbon cable should connect to the keypad. Take care – the keypad has only 9 pins! The connector should be attached so that pin 1 on the keypad (designated by the "F" on the Grayhill 20-key keypad, Part No. 86JB2), and on the left looking at the back of the keypad with the connector at the top), corresponds to pin 1 on the cable (the edge of the cable that terminates on the WildCard closest to the field header, H4). In operation, the keypad is oriented so that its connector is on the back at the bottom of the keypad.

Connecting the Display

The WildCard can accommodate liquid crystal display modules up to 4 lines by 20 characters in size. If your display still has a clear plastic cover to protect the display glass from scratches during shipment don't forget to peel it off.

The processor sends control commands and ASCII data to the display and powers its backlight through a 16-pin bus. The 16-pin single-row female connector on the ribbon cable connects directly to the display pins at the top rear of the display. The edge of the cable to connect to pin 1 of the display (nearest the display's corner) is the one aligned with pin 1 of the WildCard's header H2, and closest to the field header on the WildCard. It is also the edge closest to the keypad connector. In its standard orientation, the display connector is at the top back of the display, and the display has 4 lines and 20 characters per line.

Powering the WildCard

The Keypad/Display WildCard derives its power from the WildCard bus so it is automatically powered when plugged in. It should not be hot-inserted into an active controller however. Instead, you should turn off power, install all WildCards, then power up your system.

Adjusting the Contrast and Attaching an External Contrast Control

A contrast potentiometer is located on the board. The WildCard is shipped with a high-contrast “supertwist” LCD display that has a wide viewing angle. In most applications the contrast/viewing angle potentiometer can be left in its default setting, as shipped from the factory, with no adjustments required. If you wish to change the contrast setting, simply twist the potentiometer using a small screwdriver. While the 4x20 character displays provide good contrast over a wide viewing angle at a fixed contrast value, the contrast may need to be readjusted if the temperature changes significantly.

An additional three wires on the Keypad/Display cable bring out +5V, V_Contrast, and VEE/GND, respectively. These signals also are provided on the field header, H4. You can connect a panel-mounted potentiometer to these three signals on either connector if your application requires external control of the display’s contrast and/or viewing angle. To do so, connect the endpoints of a 10K potentiometer to +5V and VEE/GND and run its centertap to V_Contrast. Jumper J4 should be removed, and J6 set to either “Char” or “Graphics” for the display type used.

Using the General Purpose I/O

Eight lines of general purpose digital I/O are provided for your use. These lines are provided on the WildCard’s Field Header, H4, at the pin locations shown in Table 1-4.

Table 1-4 H4: Field Header

Signal	Pins	Signal
GND	1 2	+5V
+5V	3 4	V+Raw
V_Contrast	5 6	VEE/GND
+5V	7 8	/BEEPER_ON
DI_7	9 10	DI_6
DI_5	11 12	DI_4
DIO_3	13 14	DIO_2
DIO_1	15 16	DIO_0
NC	17 18	NC
NC	19 20	NC
NC	21 22	NC
NC	23 24	NC

Note:

Four general purpose digital input/outputs are brought out to pins 13-16, and four inputs to pins 9-12.

Several bytes of memory, starting at **0xC000**, are used to communicate with the WildCard. The eight digital I/O lines are mapped to a single byte to memory address **0xC00D**. The lower nibble, the four lines designated DIO_0 through DIO_3, are configurable as either inputs or outputs. The upper nibble, the four lines designated DI_4 through DI_7 are dedicated inputs. A read of the memory address returns the inputs, a write sets the outputs. The least significant bit, bit 0, of address **0xC00E** sets the direction of the configurable nibble. If bit 0 is set to 1 the nibble is configured as

outputs, if it is cleared to 0 the nibble is configured as inputs. On power up the nibble defaults to inputs. Table 1-5 diagrams the WildCard's memory map.

Table 1-5 Keypad/Display WildCard memory locations.

Address	Bit Positions	Meaning
0xC002	0	A write of 1 to the least significant bit turns on the beeper, 0 turns it off.
0xC002	1	A write of 1 turns on the backlight, 0 turns it off.
0xC00D	0-3	Either inputs or outputs
0xC00D	4-7	Digital inputs
0xC00E	0	A write of 1 configures the lower nibble of 0xC00D to all outputs; resetting to 0 configures the nibble to all inputs.

These locations are all read/write. A read of an output returns the last byte written to the output, and a read from an input returns the current state of the input pin.

When reading or writing to a single bit position you should use the kernel routines **SET.BITS** and **CLEAR.BITS** from Forth or compile the functions **SetBits()** and **ClearBits()** when using C. For example, typing from a terminal connected to your controller, to enable the lower nibble of 0xC00D as outputs and to then set DIO_2 high you would first write a 1 to the lower bit of 0xC00E as,

```
01 0xC00E 0x00 SET.BITS↵
```

and then set bit 2 high as,

```
04 0xC00D 0x00 SET.BITS↵
```

The 01 or 04 indicate the bit positions to be affected, the 0xC00E and 0xC00D indicate the 16-bit addresses written to, and the 0x00 indicates the address page. From your terminal you can just type the above lines in the terminal window and send them to your controller by pressing the “Enter” key. The arrows on the lines above indicate the action of the “Enter” key.

The **SET.BITS** command takes a bit mask and address and sets the output bits corresponding to the bits that are set in the bit mask without affecting the other output bits. The first **SET.BITS** above configures the lower nibble of the I/O port located at 0xC00D for output, and the second sets just a single bit high.

In C you would compile the C functions **SetBits()** and **ClearBits()** into your program, compile it, and download it to your controller.

Controlling the Beeper

In normal operation the beeper on the WildCard is activated in software by writing a one to the least significant bit (LSB) at address **0xC002** on the page corresponding to the modules address (which should normally be configured to 0). Because other bits of the same address are used for other functions (like the backlight), the LSB must be set and cleared using **SET.BITS** and **CLEAR.BITS** from Forth, or using **SetBits()** and **ClearBits()** in C. If the shorting bar is removed from

jumper J3 the beeper is disabled. An external device capable of sinking 40 ma. can also turn on the beeper by pulling pin 27 on header H2, or pin 8 on the field header H4, to ground. A bipolar transistor, FET or switch can be used.

Table 1-6 H2: Keypad/Display Header

Signal	Pins	Signal
KPC4	1	2 – GND
KPC3	3	4 – +5V
KPR3	5	6 – V_Contrast
KPR0	7	8 – Display_A1
KPC2	9	10 – Display_R/W
KPR1	11	12 – Display_E
KPR2	13	14 – Display_D0
KPC1	15	16 – Display_D1
KPC0	17	18 – Display_D2
+5	19	20 – Display_D3
V_Contrast	21	22 – Display_D4
VEE/GND	23	24 – Display_D5
+5	25	26 – Display_D6
/BEEPER_ON	27	28 – Display_D7
DIO_0	29	30 – LED Backlight+
DIO_1	31	32 – LED Backlight-
DIO_2	33	34 – VEE

Note:

Keypad connections are aligned along one side and the display connections are aligned along the other side. Keypad signals on odd-numbered pins 1-17 are tapped from the cable by a 10-pin single-row female connector (of which only 9 pins are actually used), and the display signals on even-numbered pins 2-32 are tapped by a 16-pin single-row female connector.

To turn on the beeper you could execute from the terminal,

```
01 0xC00C 0x00 SET.BITS←
```

and to turn it off you would enter,

```
01 0xC00C 0x00 CLEAR.BITS←
```

Using the Keypad

The keypad offers a simple yet effective means for an operator to control a computer-based instrument. By pushing a keypad button, the user shorts a “row” circuit to a “column” circuit. Your controller’s driver hardware and software detect the connection by reading the 4 rows as digital inputs, and holding them in a default “high” state with pull-up resistors. Each of the 5 columns is connected to a digital output. The processor scans each column low in turn and reads each of the row inputs to see if it has been pulled low. If it has, the processor deduces that a user is holding down the key at the intersection of that row and column.

By scanning the rows and columns, the processor can identify which key was pressed. The routines of Table 1-7 are built into the QED-Forth kernel; they scan the keypad and report which (if any) key is being depressed. Using these routines we can take any desired action based on input obtained from the keypad. Consult the C or Forth Glossaries for detailed descriptions of these routines.

Table 1-7 Keypad driver functions.

C Name	Forth Name	Function
ScanKeypad	?KEYPAD	Scans the keypad and if a key is being pressed it waits for a key <i>release</i> , returning with the key number. Calls PAUSE while waiting to enable multitasking. If no key is being pressed, it returns immediately with a -1.
ScanKeypress	?KEYPRESS	Scans the keypad returning with its current state: either -1 for no key pressed or the key number. Does not wait. ScanKeypress provides a way of monitoring the state of the keypad without “tying up” the processor while waiting for a key to be released. When using ScanKeypress , make sure that multiple calls to your program do not misinterpret a single keystroke as multiple entries from the keypad.
Keypad	KEYPAD	Waits for a key <i>release</i> and returns with the key number on key release. While waiting, it calls PAUSE to give other tasks a chance to run.

These software drivers refer to keys by numbers from 0 through 19. Key 0 is the lower right key, Key 1 is directly above it, and Key 19 is the upper left key. The keys are oriented as:

19	15	11	7	3
18	14	10	6	2
17	13	9	5	1
16	12	8	4	0

For example, if the user presses and releases the key in the upper right corner while the **KEYPAD** routine is running, the number 3 will be returned. For example, from the Forth prompt you can type in,

```
KEYPAD . ↵
```

and the controller will wait for a key press, typing the key number when it is released.

Using the Display

Your controller includes built-in software drivers for an LCD up to 4 lines by 20 characters in size. A ribbon cable connects the board to the display. The display is automatically initialized and blanked upon each reset or restart, so it is ready to use at startup.

Since displays are controlled differently depending on their size and type, your controller must be configured for the kind of display that is present. A built-in library function named **IsDisplay()** in C (or **IS.DISPLAY** in Forth) performs this configuration. Fortunately, your controller stores the display configuration information in non-volatile EEPROM that retains its data even when power is removed, so you don't have to reconfigure the board after each power-up. Both character and graphics displays are pre-configured at the factory to operate in “text mode”, so all of the sample code described here will work without further configuration on your part.

The display interface is based on a simple idea: you write the desired characters or bit-mapped graphics patterns to a display buffer in the controller's RAM, and then use the pre-coded `UpdateDisplay()` function (or `UPDATE.DISPLAY` in Forth) to transfer the contents of the buffer to the display. You control a character display by writing ASCII characters or strings to a 4 line by 20 character buffer in RAM using `$>DISPLAY` and then executing `UPDATE.DISPLAY` to make the contents visible.

Additional pre-coded routines allow you to control the cursor and write individual data and control symbols to the display. To get a feel for the capabilities of these functions, you can browse through the "Keypad/Display Interface" section of the "Categorized List of QED Library Functions" in the "Control C Glossary" Document. You can also take a look at the `INTRFACE.H` header file (in the `\FABIUS\INCLUDE\MOSAIC` directory) which contains declarations for the display control functions and macros.

Table 1-8 lists the display driver functions.

Table 1-8 Display driver functions.

C Name	Forth Name
<code>BufferPosition()</code>	<code>BUFFER.POSITION</code>
<code>CharsPerDisplayLine()</code>	<code>CHARS/DISPLAY.LINE</code>
<code>CharToDisplay()</code>	<code>CHAR>DISPLAY</code> and <code>BYTES>DISPLAY</code>
<code>ClearDisplay()</code>	<code>CLEAR.DISPLAY</code>
<code>CommandToDisplay()</code>	<code>COMMAND>DISPLAY</code>
<code>DisplayBuffer()</code>	<code>DISPLAY.BUFFER</code>
<code>DisplayOptions()</code>	<code>DISPLAY.OPTIONS</code>
<code>DISPLAY_HEAP</code>	<code>DISPLAY.HEAP</code>
<code>GARRAY_XPFA</code>	<code>GARRAY.XPFA</code>
<code>InitDisplay()</code>	<code>INIT.DISPLAY</code>
<code>IsDisplay()</code>	<code>IS.DISPLAY</code>
<code>IsDisplayAddress()</code>	<code>IS.DISPLAY.ADDRESS</code>
<code>LinesPerDisplay()</code>	<code>LINES/DISPLAY</code>
<code>PutCursor()</code>	<code>PUT.CURSOR</code>
<code>StringToDisplay()</code> and <code>STRING_TO_DISPLAY()</code>	<code>\$>DISPLAY</code>
<code>UpdateDisplay()</code>	<code>UPDATE.DISPLAY</code>
<code>UpdateDisplayLine()</code>	<code>UPDATE.DISPLAY.LINE</code>
<code>UpdateDisplayRam()</code>	<code>(UPDATE.DISPLAY)</code>

While there are a great many functions associated with the display, allowing you lots of flexibility in using the display, you really need only a few to make it work.

Programming the Display in Forth

The operating system maintains an 80 character buffer whose base extended address (or *xaddress*) is returned by the routine,

```
DISPLAY.BUFFER ( -- xaddr )
```

The offset from the start of this buffer to a specified line and character position is returned by the routine,

```
BUFFER.POSITION ( line#\char# -- buffer.offset )
```

The user can write ASCII characters into this buffer using standard operators such as **C!** or **CMOVE**, or with the assistance of the handy utility routine

```
$>DISPLAY ( x$addr\line#\char# -- )
```

which is pronounced “string-to-display” (in Forth, \$ is often used to represent a string).

\$>DISPLAY moves the string starting at x\$addr to the buffer starting at the specified line number (0, 1, 2, or 3) and character position (0 through 19) in the display buffer. This routine moves only as many characters as will fit on the specified line. Executing **\$>DISPLAY** modifies the contents of the buffer but does not alter the display.

Placing the appropriate line number (0, 1, 2, or 3) on the stack and executing

```
UPDATE.DISPLAY.LINE ( line# -- )
```

transfers the specified line’s contents from the buffer to the display. Executing **UPDATE.DISPLAY** transfers all of the lines in the display buffer to the display.

Let’s try an example. Type in the following definition:

```
: SHOW.MESSAGE ( -- )
  CLEAR.DISPLAY
  “ My favorite color is” 0 0 $>DISPLAY
  “ purple.” 1 0 $>DISPLAY
  UPDATE.DISPLAY
;
```

SHOW.MESSAGE first executes **CLEAR.DISPLAY** which clears the display and fills the **DISPLAY.BUFFER** with ASCII blanks. The next two lines in the definition specify the contents of the top two lines of the display. The characters between the quotation marks specify a string to be moved to the display buffer, and the two numbers immediately preceding **\$>DISPLAY** are the line number (numbered 0 through 3) and the character position (numbered 0 through 19) to which the string is moved. If you execute

```
SHOW.MESSAGE←
```

the message will appear on the display.

To modify only the line reporting the color “purple” without changing the rest of the display, you could execute from the terminal,

```
DISPLAY.BUFFER 1 0 BUFFER.POSITION XN+ 20 BLANK←
“ blue.” 1 0 $>DISPLAY←
```

```
1 UPDATE.DISPLAY.LINE←
```

The first command blanks line#1 in the display, eliminating the prior contents. **BUFFER.POSITION** calculates the offset to the start of line 1, and **XN+** adds this offset to the **DISPLAY.BUFFER** base address to yield the start address of line# 1 in the buffer. **BLANK** then blanks the 20 characters on the line in the buffer. The following command line moves the string “blue.” to the display buffer, and **UPDATE.DISPLAY.LINE** writes the new contents of line 1 to the display. Note that the top line of the display is unchanged.

We could have avoided the need for the **BLANK** command by making the new “color” string as long or longer than the original string on line 1. For example, the following two commands have the same effect as three commands above:

```
“ blue.          ” 1 0 $>DISPLAY←
1 UPDATE.DISPLAY.LINE←
```

The extra trailing spaces in the string ensure that none of the prior string remains in the buffer.

To clear the display, fill the display buffer with blanks, and home the cursor to the upper left corner of the display, simply type

```
CLEAR.DISPLAY←
```

and of course executing **SHOW.MESSAGE** again replaces the original message on the display.

The routine **INIT.DISPLAY** initializes the display so that it is ready to accept characters, homes the cursor to the upper left position, and blanks the screen, cursor, and display buffer. This routine is executed upon every processor reset or restart. **INIT.DISPLAY** also initializes the system variable **LINES/DISPLAY** to 4 and the system variable **CHARS/DISPLAY.LINE** to 20. The values of these variables may be modified by the programmer to accommodate different sized displays up to a maximum of 80 characters. For example, to interface a 2 line by 16 character display, execute

```
DECIMAL 2 LINES/DISPLAY ! 16 CHARS/DISPLAY.LINE !←
```

Throughout the above examples you probably noticed that no cursor was visible on the display. This is because **INIT.DISPLAY** also turns the cursor off. You have full control over the cursor, however. The routine

```
DISPLAY.OPTIONS ( display.on?\cursor.on?\cursor.blinking? -- )
```

sets the cursor and display state based on the three flags passed to it on the stack. The first flag, called **display.on?**, specifies whether the display is enabled or disabled. This feature can be used to flash the display by rapidly enabling and disabling the display via successive calls to **DISPLAY.OPTIONS**. The second flag, called **cursor.on?**, specifies whether the cursor is visible as an underbar at the current cursor position. The top flag on the stack, called **cursor.blinking?**, specifies whether the cursor is visible as a blinking box obscuring the current character position. The default condition is: display enabled, cursor off, cursor not blinking. This condition applies after a reset, restart, or execution of **INIT.DISPLAY**. Try passing different flag combinations to **DISPLAY.OPTIONS** to see what the results are.

Several lower level utilities are available to the programmer. For example, the routine

```
PUT.CURSOR ( line#\character# -- )
```

places the cursor at the specified location on the display. Note that whether or not the cursor is visible depends on the configuration set by **DISPLAY.OPTIONS**. Once the cursor has been placed, each succeeding character written to the display appears at the cursor location and causes the cursor position to be incremented by 1. Please note, however, that in many displays the cursor does not advance smoothly as one might expect from the end of one line to the start of the following line! Please test your routines carefully when using these low level utilities to control the display.

To send a single character directly to the display at the current cursor position, bypassing the display buffer, execute

```
CHAR>DISPLAY ( char -- )
```

This automatically increments the cursor position (but note that the cursor may skip to the start of an unexpected line after the end of a line is reached). **CHAR>DISPLAY** does not update the contents of the **DISPLAY.BUFFER**.

To send a command character directly to the display, execute

```
COMMAND>DISPLAY ( byte -- )
```

Consult the display data sheet appendix to this document to determine the numerical value associated with each valid command. For example, the command byte that clears the display is 0x01.

The following routine uses both the keypad and display, showing on the display the key number of a pressed key:

```
: TEST.KEYPAD ( -- )
  INIT.DISPLAY \ Initialize the display
  BEGIN
    PAUSE.ON.KEY \ allows you to type a CR from the terminal to bail out of the loop
    " You pressed key#:" 0 0 $>DISPLAY \ write to buffer
    KEYPAD \ get the key number ( -- key# )
    S>D <# # # #> DROP 1XN- ( -- x$addr ) \ convert key# to a string
    1 0 $>DISPLAY ( -- ) \ write key# to buffer
    UPDATE.DISPLAY \ write to display
  AGAIN
;
```

After executing **TEST.KEYPAD**, the identifier of each key that you press is displayed as a 2-digit number (in the current **BASE**) on the LCD display. Type a carriage return at the terminal and press one final key on the keypad to exit the routine.

The following code provides a more complete example. It allows the user to test the display by putting a grid of "X"s on the display, one "X" for each keypad button. Then as the user presses each key the corresponding "X" is removed. It also produces a key click on each key release, showing how to make beeps of a particular duration, and shows how to turn on the backlight. You can download the following code after any coldstart.

```
DOWNLOAD.MAP
0x04 USE.PAGE
```

```
ANew Display.Keypad.Tester
```

```
DECIMAL \ Numbers are interpreted as decimal unless preceeded with "0x";
        \ if preceeded with "0x" they are interpreted as hexadecimal.
```

```

\ A constant is defined for the address of the WildCard. Page 0x00
\ corresponds to a WildCard plugged into Moduel Port 0 with no jumpers set.
\ A WildCard plugged directly into a QCard is plugged into Module Port 0.
\ Ordinarily, WildCards can be configured for module addresses 0 to 7;
\ but the Keypad/Display WildCard MUST be set to module address 0 for the
\ built-in software drivers to work. All WildCards use 16-bit addresses
\ starting at 0x0C00. The 16-bit value at address 0xC000 is a security key;
\ bits at address 0xC002 control the beeper (the lsb) and backlight (the next bit);
\ a bit at 0xC00E configures the I/O direction of the lower nibble of 0xC00D;
\ and 0xC00D provides 8 bits of general purpose user I/O.
\
0xC002 0x00 XCONSTANT Beeper/BacklightAddr \ WildCard MUST be located on page 0

: >Beeper ( u -- | u is the number of microseconds to hold the beeper on)
\ An audible key click is made by actuating the beeper for a short period.
\ 1 millisecond is just enough time to give a good solid click, less time gives
\ a weaker click. Beeps up to 65.535 milliseconds are possible with this routine.
0x01 Beeper/BacklightAddr SET.BITS \ the LSB of location C002\00 controls beeper
MICROSEC.DELAY \ delays for u microseconds
0x01 Beeper/BacklightAddr CLEAR.BITS \ we use SET.BITS and CLEAR.BITS so as to not
; \ affect any other bits at the address

: Show.Keypad.Test.Screen ( -- )
\ Shows X's for each keypad button position. As each button is pressed
\ an X in the corresponding location disappears.
" ***** XXXXX *" 0 0 $>DISPLAY \ Stores the string at the row\col
" Press keys XXXXX *" 1 0 $>DISPLAY \ position in the display buffer
" to test. XXXXX *" 2 0 $>DISPLAY \ UPDATE.DISPLAY then transfers the
" ***** XXXXX *" 3 0 $>DISPLAY \ buffer to the screen
UPDATE.DISPLAY
;

: Show.Exit.Screen ( -- )
\ Display screen to show after the test is done.
" Congratulations ! " 0 0 $>DISPLAY \ Stores the string at the row\col
" " 1 0 $>DISPLAY \ position in the display buffer
" All the keys work " 2 0 $>DISPLAY \ UPDATE.DISPLAY then transfers the
" as they should. " 3 0 $>DISPLAY \ buffer to the screen
UPDATE.DISPLAY
;

: Key>Row\Col ( key.num -- row\col )
\ Converts a keypad button number in the range 0-19 to
\ row and column numbers, 0-3 for rows, and 0-4 for columns.
\ The keypad is laid out as:
\ Row Col: 0 1 2 3 4
\ 0 19 15 11 7 3
\ 1 18 14 10 6 2
\ 2 17 13 9 5 1
\ 3 16 12 8 4 0
4 /MOD \ row.offset/col.offset
3 ROT - \ col.offset/row
4 ROT - \ row/col
;

: KEY>DISPLAY.ADDR ( key.num -- char.addr.in.display.buffer )
Key>Row\Col 13 + \ increment col# to start of array of X's on the screen
BUFFER.POSITION \ converts row\col to a display buffer offset
DISPLAY.BUFFER \ leave display buffer address on stack
ROT XN+ \ bring offset to top and add it to display buffer address
;

: KEYPAD.TEST ( -- )
\ This routine waits for the user to press each key. As each key is pressed it
\ removes an "X" from the display at the key position. After all the keys are
\ pressed this routine returns. We'll use the bit positions in a local variable to
\ keep track of which keys are pressed, setting a different bit for each different
\ key pressed.

```

```

0x0000          \ for keys numbered 0-15, start with all bits cleared
0xFFFF         \ for keys numbered 16-19, start only with four bits cleared
LOCALS{ &msb.keys &lsb.keys } \ hold the key positions in local variables
0x02 Beeper/BacklightAddr SET.BITS \ Turn on the backlight
4 20 TRUE TRUE TRUE \ Define display to have 4 rows, 20 columns, text mode,
IS.DISPLAY      \ a char.display, and to use hitachi controller chip
INIT.DISPLAY    \ Initializes the display, reserves the DISPLAY.BUFFER in RAM
                \ and calls CLEAR.DISPLAY.
Show.Keypad.Test.Screen \ Put our instructions to the user on the screen
BEGIN
  PAUSE.ON.KEY    \ Allows premature exit with a CR from the serial port
&msb.keys &lsb.keys XOR \ Continue until all bits are set
  WHILE
    KEYPAD        \ loops on the keypad & returns the number of pressed key
    1000 >Beep    \ makes a key click to provide audible feedback
    DUP KEY>DISPLAY.ADDR 20 -ROT C! \ place ascii blank into key's location in
    UPDATE.DISPLAY \ the display buffer and update the display to show it
    DUP 16 <      \ Is the key number less than 16?
    IF            \ If so, set a bit in &lsb.keys
      1 SWAP SCALE &lsb.keys OR TO &lsb.keys
    ELSE          \ If not, subtract 16 and set a bit in &msb.keys
      16 - 1 SWAP SCALE &msb.keys OR TO &msb.keys
    ENDIF
  REPEAT
  Show.Exit.Screen
;

```

Programming the Display in C

Programming the display in C is very similar to programming it in Forth. Table 1-7 and Table 1-8 provide the corresponding function names for the keypad and display drivers.

Compiling and Downloading the HELLO.C Program

The source code for an example program discussed in the remainder of this chapter is in the **HELLO.C** file in the \FABIUS\QEDCODE directory. Here is a brief summary of how to compile the program:

1. Open the Msoaic (TextPad) editor by double-clicking on its icon.
2. Choose “Open” from the “File” menu, and open the file named **HELLO.C** in the \FABIUS\QEDCODE directory.
3. Click on the editor’s “MAKE” icon to compile the source code file and automatically generate a download file named **HELLO.TXT** ready to be sent to your controller.
4. Now that the program is compiled, you can download it to the controller by entering the “Terminal” program by clicking toolbar icon. Or, if the Terminal is already active, enter it by clicking in its window.
5. Choose “Send Text File” from the Terminal menu, change the directory to \FABIUS\QEDCODE by clicking on the appropriate folders, and double click on the **HELLO.TXT** filename to transfer the download file to the controller.
6. You will see a hexadecimal download in Motorola S2-record format scroll across your terminal window. At the end of the download you will see some brief Forth function definitions;

these are telling the onboard operating system the names of the individual functions and variables so that we can interactively execute the functions in the `HELLO.C` file.

7. To run the program, simply type `main` from the terminal window. Several things should happen:

When you run the program you should see the message,

```
Hello world!
```

printed in your terminal window. You should see the following message on your LCD display:

```
Welcome!...Press any  
keypad button to see  
how the display and  
keypad work together
```

If you press any button on the keypad, you should see the message

```
I'd rather be...
```

on the first line, followed by a message on the second line that varies depending on the column of the keypad button that you choose. Try it out. To terminate the test, push a button in the left-most column of the keypad; you'll see the message

```
I'd rather be...  
Done with this test.
```

Now if you type carriage returns from your terminal, the QED-Forth monitor will respond with the “ok” prompt, meaning that it is ready for the next command.

You can type:

```
main↵
```

from your terminal any time you like, and the `main` function as defined in the latest download file will be executed.

Writing to the LCD Display

Let's take a look at the definition of the `ShowMessage()` function whose source code can be found near the bottom of the `HELLO.C` file. This function writes a message to the Liquid Crystal Display (LCD). The definition of the function is:

```
_Q void ShowMessage(void)
{
    STRING_TO_DISPLAY("Welcome! Press any ", 0, 0);
    STRING_TO_DISPLAY("keypad button to see", 1, 0);
    STRING_TO_DISPLAY("how the display and ", 2, 0);
    STRING_TO_DISPLAY("keypad work together", 3, 0);
    UpdateDisplay();
}
```

The `_Q` keyword declares this function as one that can be interactively called, and the `void` keywords tell the compiler that this function does not expect any input parameters and does not return a value. Each of the first four lines in the definition specifies the contents of a line on the display. The `STRING_TO_DISPLAY()` macro expects three input parameters: a string pointer, the line number, and the character position. It writes the specified string into the display buffer (located in the con-

troller's RAM) starting at the specified line and character position. Note that each of the first four statements in the definition represents one line of the message, and the string is displayed starting at character position 0 (the left-most position) on each line. Lines are numbered 0 through 3 on a character display. Character positions are numbered 0 through 19 on a character display.

The fifth statement in `ShowMessage()` calls the `UpdateDisplay()` function which writes the contents of the display buffer to the display.

To show the welcoming message on the display without printing a message to the terminal or running the keypad demonstration, type from your terminal:

```
ShowMessage( )←
```

followed by a carriage return. Be sure to type at least one space after the `(` character. You should see the welcoming message appear on your LCD display. You will also see at your terminal a line of text that summarizes the return value of the function in several formats (decimal, hexadecimal, and floating point), followed by the “ok” prompt. Because the `ShowMessage()` function does not return anything, the return value summary is not useful here. The “ok” prompt indicates that controller has successfully called the function and is now ready to execute another command.

For more information, consult the detailed descriptions of the `STRING_TO_DISPLAY()` macro and the `UpdateDisplay()` function in the “Control C Glossary” in your documentation package.

Keypad

Now let's take a look at the `ManageKeypad()` routine whose source code appears just before `ShowMessage()` in the `HELLO.C` file:

```
_Q void ManageKeypad(void)
{ int done = 0;
  while(!done)
  { switch((int) Keypad() / 4)
    { case 0: idRatherBe("Sky Diving"); break; // col 0
      case 1: idRatherBe("Traveling"); break; // col 1
      case 2: idRatherBe("Watching TV"); break; // col 2
      case 3: idRatherBe("Eating"); break; // col 3
      case 4: idRatherBe("Done with this test.");
        done = 1; break; // col 4
    }
  }
}
```

As described earlier, the `_Q` declaration tags the function as one that will be interactively callable from QED-Forth to speed debugging. The void keywords tell the compiler that this function does not return a value, and does not expect any input parameters. The first line declares an automatic (stack-based) variable named “done” and initializes it to zero. The next line is a while statement that runs until the “done” flag is true (nonzero). The body of the while loop is a switch statement that calls the `Keypad()` function which waits for a keypress and returns the index of the selected key. This index is divided by 4 using truncating integer arithmetic as indicated by the `(int)` cast to calculate the selected keypad column number. The column number is used by the switch statement to select which message is displayed on the screen. If column 4 at the left of the keypad is selected, the “done” flag is set to true and the while statement terminates.

To interactively execute the function, simply type at your terminal:

```
ManageKeypad( )←
```

followed by a carriage return; remember to type at least one space after the (character. The function is now waiting for you to press a button on the keypad; once you do, you'll see the message:

```
I'd rather be...  
<selected string goes here>
```

on the display. The routine will continue to run until you choose a key in the leftmost column of the keypad. Then you will see the printed summary of the routine's return value at your terminal, followed by QED-Forth's "ok" prompt. The return value summary is not meaningful in this case because the `ManageKeypad()` function does not return a value.

Keypad/Display WildCard Schematics

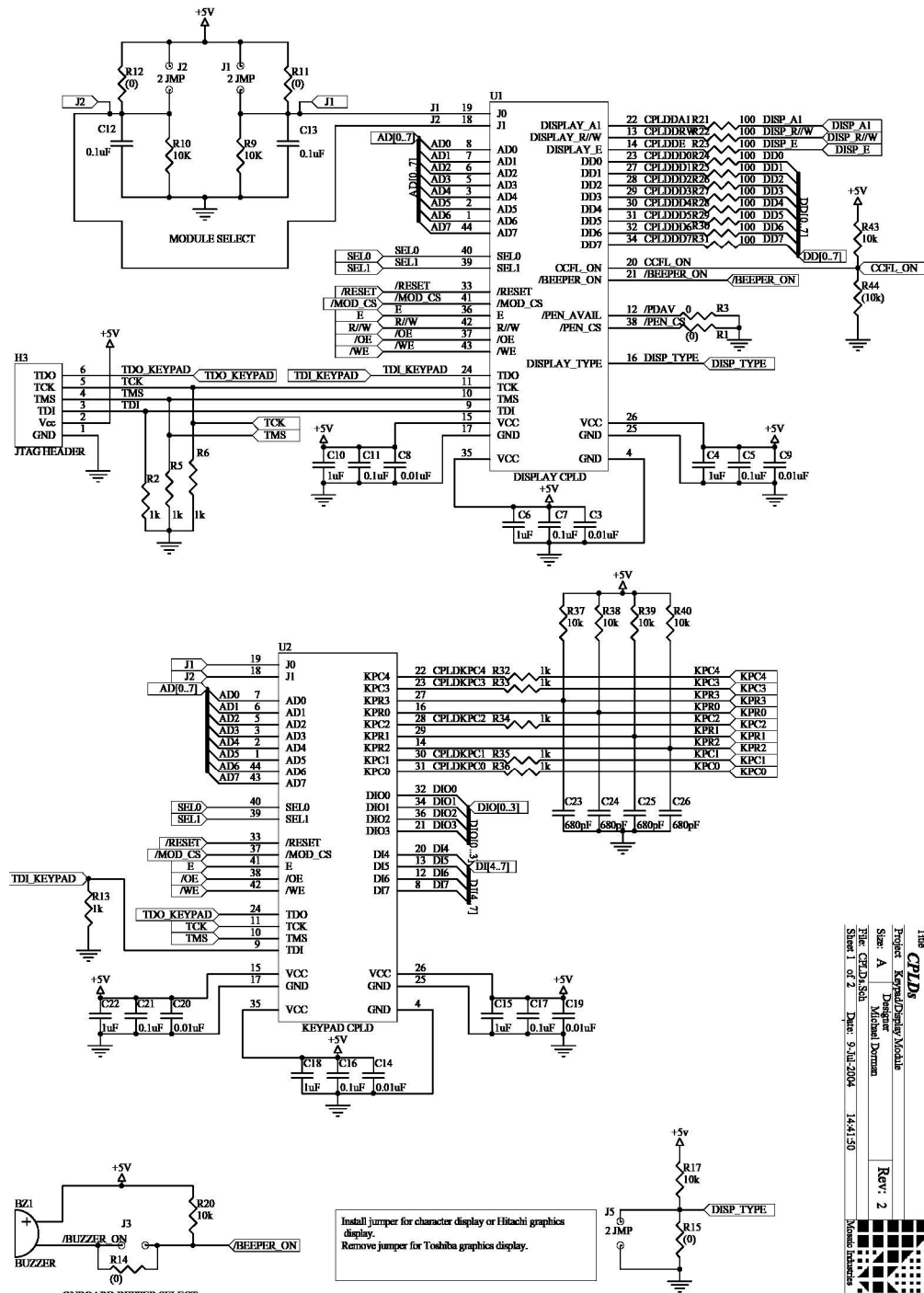


Figure 1-2 Keypad/Display WildCard Schematic Page 1.

Appendix A: Display Specifications

Contents

1. SPECIFICATIONS

- 1.1 Features
- 1.2 Mechanical Specifications
- 1.3 Absolute Maximum Ratings
- 1.4 DC Electrical Characteristics
- 1.5 Optical Characteristics
- 1.6 Backlight Characteristics

2. MODULE STRUCTURE

- 2.1 Counter Drawing
- 2.2 Interface Pin Description
- 2.3 Timing Characteristics
- 2.4 Display Command
- 2.5 Character Pattern

3. QUALITY ASSURANCE SYSTEM

- 3.1 Quality Assurance Flow Chart
- 3.2 Inspection Specification

4. RELIABILITY TEST

- 4.1 Reliability Test Condition

5. PRECAUTION RELATING PRODUCT HANDLING

- 5.1 Safety
- 5.2 Handling
- 5.3 Storage
- 5.4 Terms of Warranty

Note : For detailed information please refer to IC data sheet : ST7066U, ST7063

1. SPECIFICATIONS

1.1 Features

Item	Standard Value
Display Type	20 *4 Characters
LCD Type	STN,Gray, Transflective, Positive, Normal Temp.
Driver Type	1/16 Duty , 1/5Bias
Viewing Direction	6 O'clock
Backlight	Yellow-Green LED B/L
Weight	70.0g
Other	—

1.2 Mechanical Specifications

Item	Standard Value	Unit
Outline Dimension	98.0 (L) * 60.0 (w) *14.0 (H)(Max)	mm
Viewing Area	76.0 (L) * 25.2 (w)	mm
Active Area	70.4 (L) *20.8 (w)	mm
Dot Size	0.55 (L) * 0.55 (w)	mm
Dot Pitch	0.60(L) * 0.60(w)	mm

1.3 Absolute Maximum Ratings

Item	Symbol	Condition	Min.	Max.	Unit
Power Supply Voltage	V _{DD}	—	-0.3	7.0	V
LCD Driver Supply Voltage	V _{LCD}	—	V _{DD} -10.0	V _{DD} +0.3	V
Input Voltage	V _{IN}	—	-0.3	V _{DD} +0.3	V
Operating Temperature	T _{OP}	—	0	50	°C
Storage Temperature.	T _{ST}	—	-20	70	°C
Humidity	H _D	—	-	90	%RH

1.4 DC Electrical Characteristics

$$V_{DD} = 5.0 \text{ V} \pm 10\% , V_{SS} = 0\text{V} , T_a = 25^\circ\text{C}$$

Item	Symbol	Condition	Min.	Typ.	Max.	Unit
Logic Supply Voltage	V_{DD}	—	4.5	5	5.5	V
"H" Input Voltage	V_{IH}	—	$0.7 V_{DD}$	—	V_{DD}	V
"L" Input Voltage	V_{IL}	—	-0.3	—	0.6	V
"H" Output Voltage	V_{OH}	$I_{OH} = -0.1\text{mA}$	3.9	—	V_{DD}	V
"L" Output Voltage	V_{OL}	$I_{OL} = 0.1\text{mA}$	—	—	0.4	V
Supply Current	I_{DD}	$V_{DD} = 5.0 \text{ V}$	—	2.0	3.0	mA
LCD Driver Voltage	V_{OP}	0°C	—	—	—	V
		25°C^*1	—	4.7	—	
		50°C	—	—	—	

Note*1: The VOP test point is $V_{DD} \sim V_O$

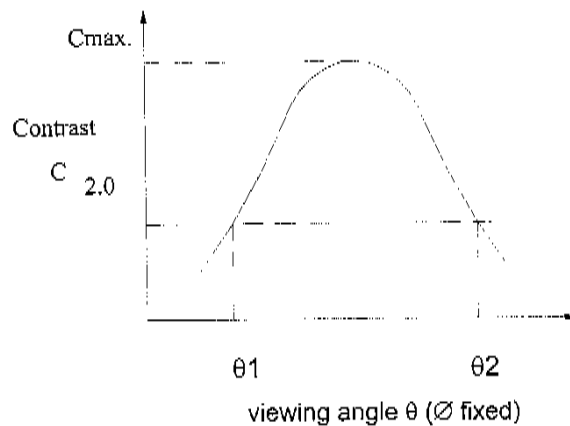
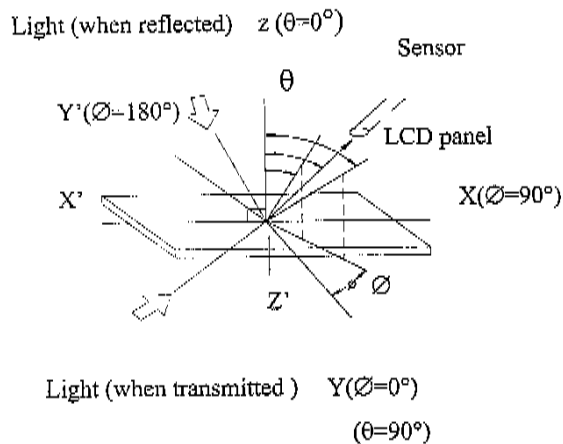
1.5 Optical Characteristics

$$\text{LCD Panel : } 1/16\text{Duty} , 1/4\text{Bias} , V_{LCD} = 4.6\text{V} , T_a = 25^\circ\text{C}$$

Item	Symbol	Conditions	Min.	Typ.	Max.	Reference
View Angle	θ	$C \geq 2.0, \phi = 0^\circ$	40°	—	—	Notes 1 & 2
Contrast Ratio	C	$\theta = 25^\circ, \phi = 0^\circ$	4	6	—	Note 3
Response Time(rise)	t_r	$\theta = 25^\circ, \phi = 0^\circ$	—	200 ms	300 ms	Note 4
Response Time(fall)	t_f	$\theta = 25^\circ, \phi = 0^\circ$	—	150ms	225 ms	Note 4

Note 1: Definition of angles θ and ϕ

Note 2: Definition of viewing angles θ_1 and θ_2

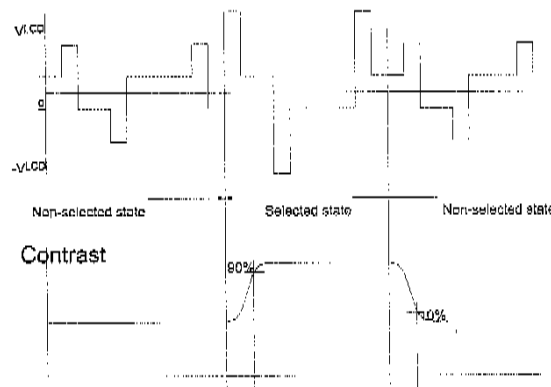
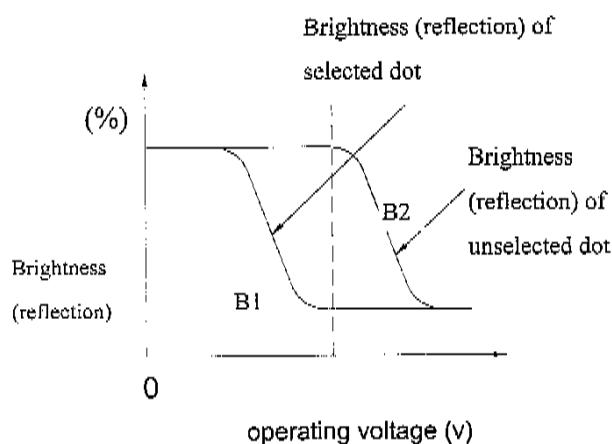


Note : Optimum viewing angle with the naked eye and viewing angle θ at C_{max} . Above are not always the same

Note 3: Definition of contrast C

Note 4: Definition of response time

$$C = \frac{\text{Brightness (reflection) of unselected dot (B2)}}{\text{Brightness (reflection) of selected dot (B1)}}$$



Note: Measured with a transmissive LCD panel which is displayed 1 cm^2

V_{LCD} : Operating voltage f_{FRM} : Frame frequency
 t_r : Response time (rise) t_f : Response time (fall)

t_r : Response time (rise) t_f : Response time (fall)

1.6 Backlight Characteristic

LCD Module with LED Backlight

•Maximum Ratings

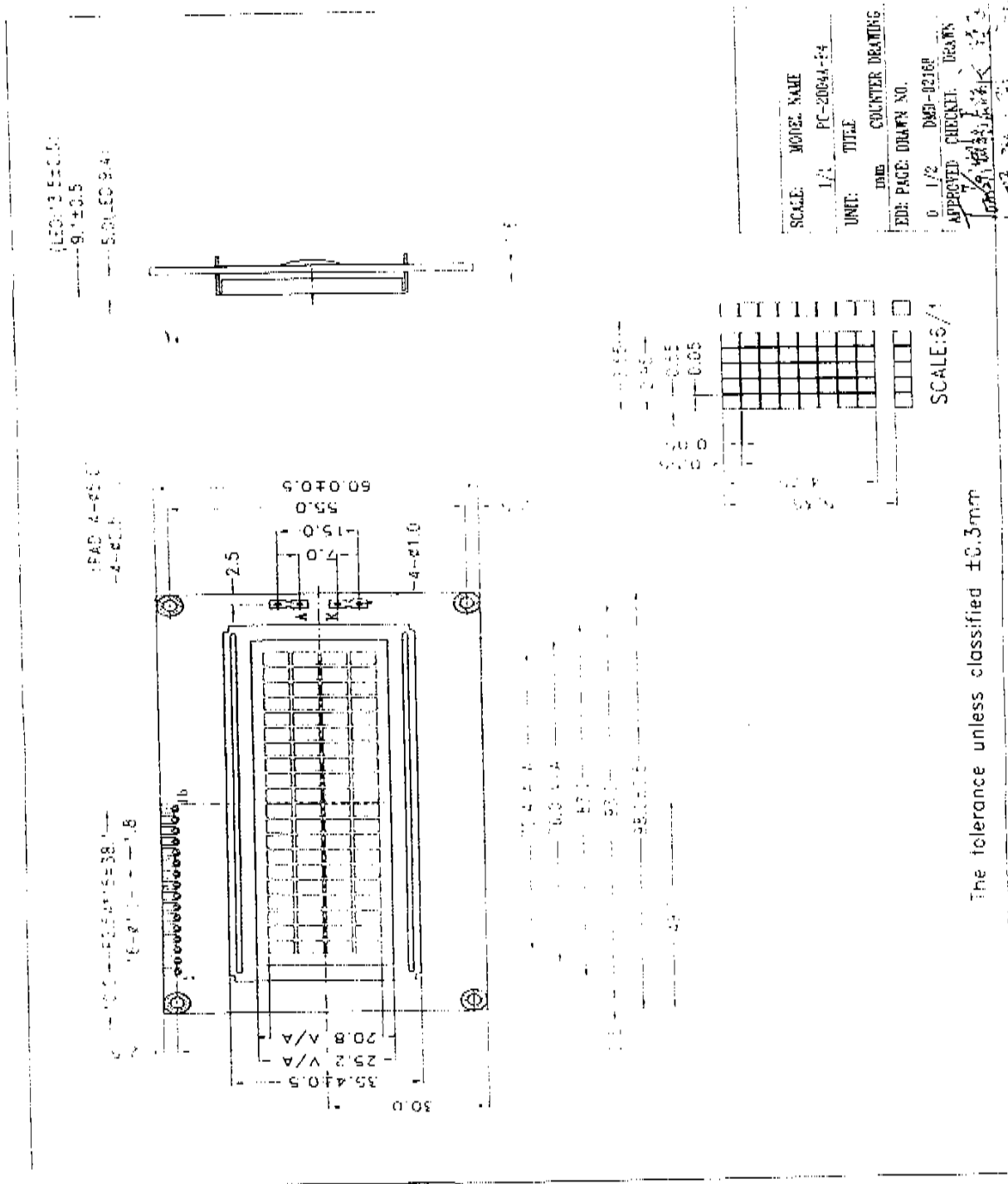
Item	Symbol	Conditions	Min.	Max.	Unit
Forward Current	IF	Ta =25°C	-	650	mA
Reverse Voltage	VR	Ta =25°C	-	8	V
Power Dissipation	PO	Ta =25°C	-	3.0	W
Operating Temperature	T _{OP}	-	-20	70	°C
Storage Temperature	T _{ST}	-	-40	80	°C
Solder Temp. for 3 Second	-	-	-	260	°C

•Electrical Ratings

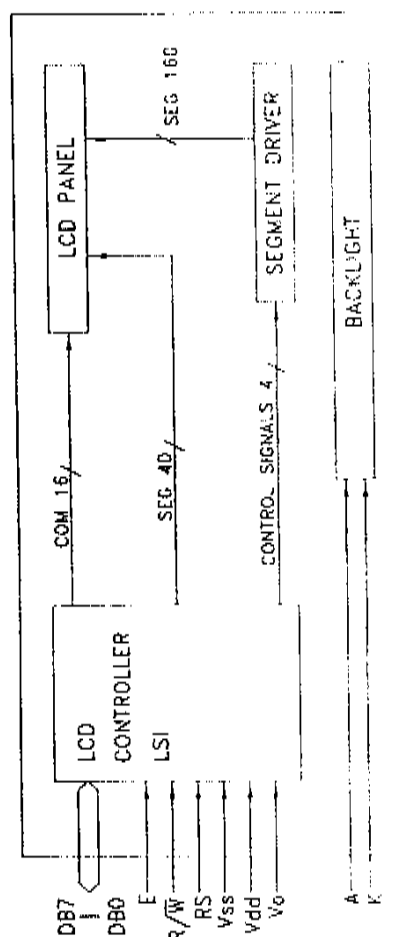
Item	Symbol	Condition	Min.	Typ.	Max.	Unit
Forward voltage	V _F	I _F =260mA	-	4.2	4.6	V
Reverse current	I _R	V _R =8V	-	-	0.2	mA
Luminous intensity	I _V	I _F =260mA	200	250	-	cd/m ²
Wavelength	λ_p	I _F =260mA	571	-	576	nm
Color	Yellow-Green					

2. MODULE STRUCTURE




2.1 Counter Drawing



1	V _{SS}
2	V _{DD}
3	V _O
4	RS
5	R/W
6	E
7	DB0
8	DB1
9	DB2
10	DB3
11	DB4
12	DB5
13	DB6
14	DB7
15	A
16	K



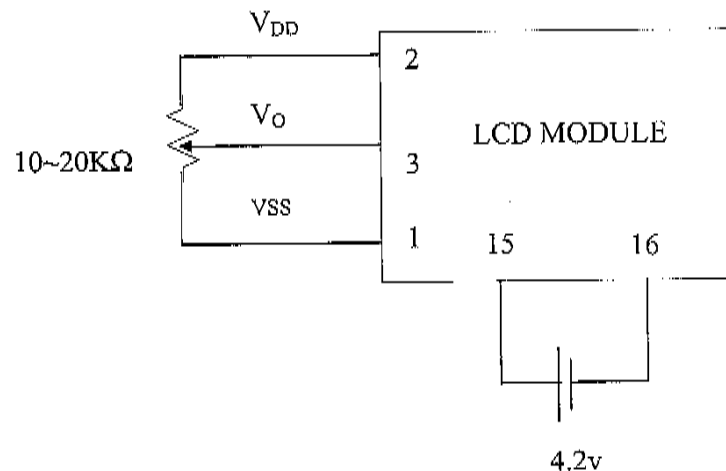
SCALE:	MODEL NAME
NO SCALE	PC-2004A-P4
UNIT:	TITLE
NO UNIT	COUNTER DRAWING
ED: PAGE	DRAWN NO.
0 2/2	DWD-02168

Q	2/2	DML-00100
APPROVED CHECKED DRAWN		
  		
2003.08.22 14:10 2003.08.22 14:10 2003.08.22 14:10		

2.2 Pin Description

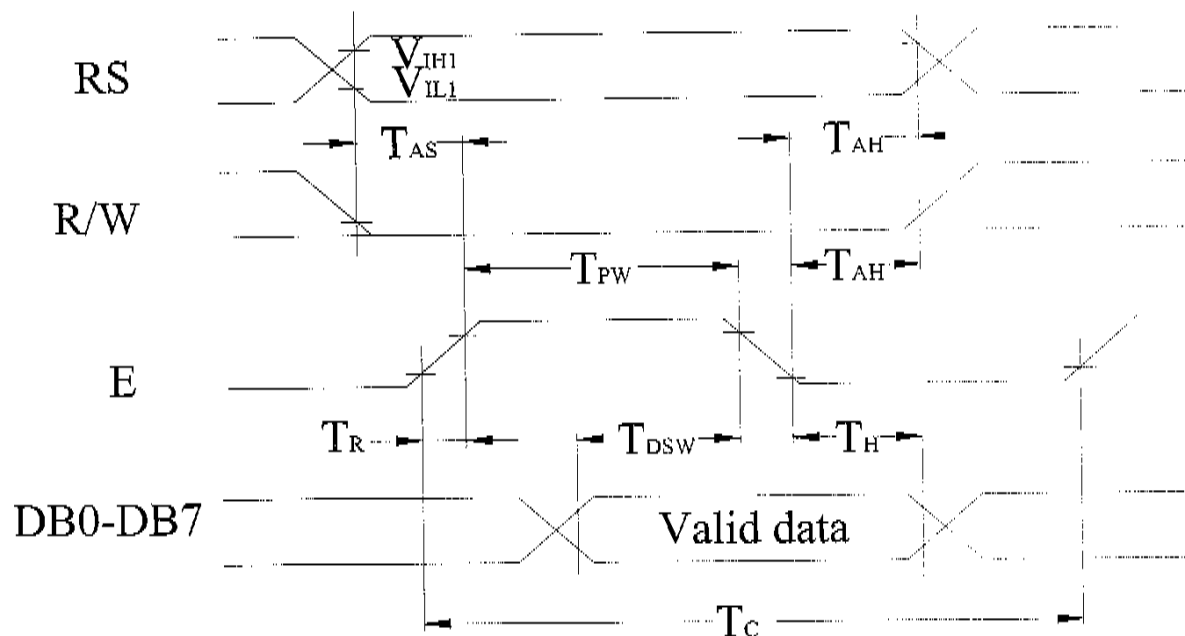
Pin No.	Symbol	Signal Description
1	VSS	Signal ground (GND)
2	VDD	Power Supply (5 V)
3	Vo	Operating voltage (LCD Driver)
4	RS	Register Selection input High = Data register Low = Instruction register (for write) Busy flag address counter (for read)
5	R/W	Read/Write signal input is used to select the read/write mode High = Read mode, Low = Write mode
6	E	Start enable signal to read or write the data
7~10	DB0 ~ DB3	Four low order bi-directional three-state data bus lines. Use for data transfer between the MPU and the LCD module. These four are not used during 4-bit operation.
11~14	DB4 ~ DB7	Four high order bi-directional three-state data bus lines. Used for data transfer between the MPU and the LCD module. DB7 can be used as a busy flag.
15	A	Power voltage supply for LED(+)
16	K	Power voltage supply for LED(-)

Contrast Adjust

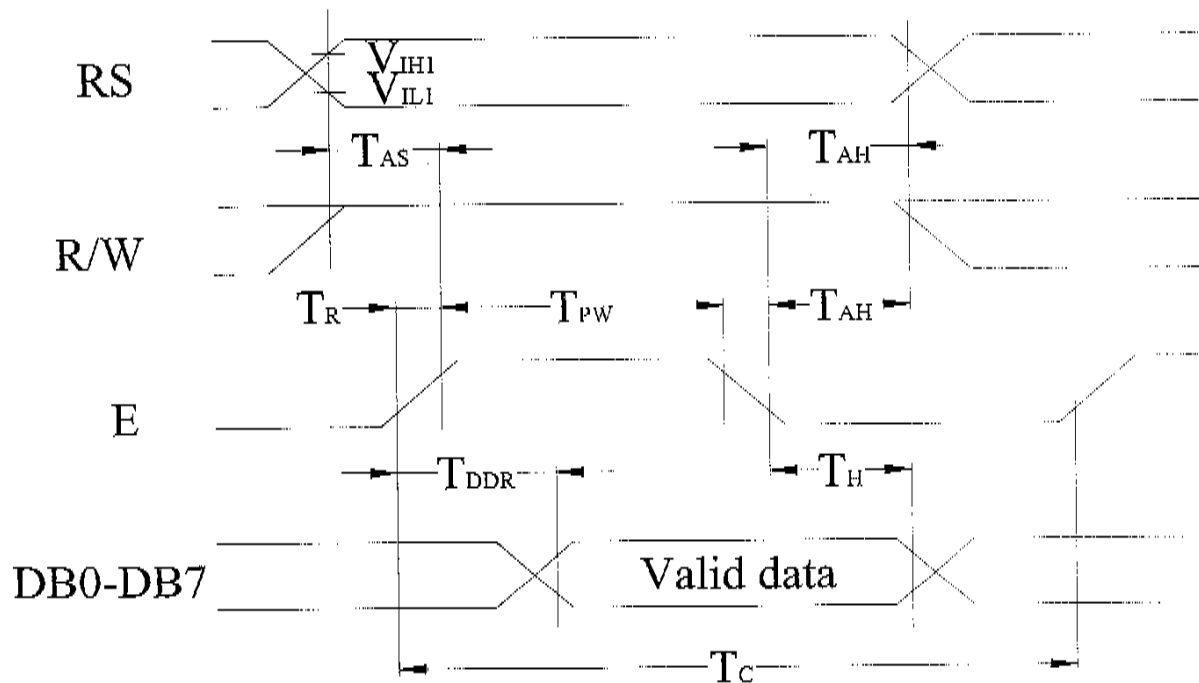


2.3 Timing Characteristics

- Writing data from MPU to ST7066U



- Reading data from ST7066U to MPU



• Write Mode (Writing data from MPU to ST7066U)

(VDD = +5V \pm 10%, Ta=25°C)

Symbol	Characteristics	Test Condition	Min.	Typ.	Max.	Unit
T _C	Enable Cycle Time	Pin E	1200	-	-	ns
T _{PW}	Enable Pulse Width	Pin E	140	-	-	ns
T _R , T _F	Enable Rise / Fall Time	Pin E	-	-	25	ns
T _{AS}	Address Setup Time	Pins: RS, RW, E	0	-	-	ns
T _{AH}	Address Hold Time	Pins: RS, RW, E	10	-	-	ns
T _{DSW}	Data Setup Time	Pins: DB0~DB7	40	-	-	ns
T _H	Data Hold Time	Pins: DB0~DB7	10	-	-	ns

• Read Mode (Reading data from ST7066U to MPU)

(VDD = +5V \pm 10%, Ta=25°C)

Symbol	Characteristics	Test Condition	Min.	Typ.	Max.	Unit
T _C	Enable Cycle Time	Pin E	1200	-	-	ns
T _{PW}	Enable Pulse Width	Pin E	140	-	-	ns
T _R , T _F	Enable Rise / Fall Time	Pin E	-	-	25	ns
T _{AS}	Address Setup Time	Pins: RS, RW, E	0	-	-	ns
T _{AH}	Address Hold Time	Pins: RS, RW, E	10	-	-	ns
T _{DDR}	Data Setup Time	Pins: DB0~DB7	-	-	100	ns
T _H	Data Hold Time	Pins: DB0~DB7	10	-	-	ns

2.4 Display Command

Instructions	Instruction Code										Description	Description Time (270KHz)
	RS	R/W	DB 7	DB 6	DB 5	DB 4	DB 3	DB 2	DB 1	DB 0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Write "20H" to DDRAM. and set DDRAM address to "00H" from AC.	1.52ms
Return Home	0	0	0	0	0	0	0	0	1	x	Set DDRAM address to "00H" from AC and return cursor to it's original position if shifted. The contents of DDRAM are not changed.	1.52ms
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.	37 μ s
Display ON/OFF	0	0	0	0	0	0	1	D	C	B	D=1 : entire display on C=1 : cursor on B=1 : cursor position on	37 μ s
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	x	x	Set cursor moving and display shift control bit, and the direction, without changing of DDRAM data.	37 μ s
Function Set	0	0	0	0	1	DL	N	F	x	x	DL: interface data is 8/4 bits NL: number of line is 2/1 F: font size is 5 \times 11/5 \times 8	37 μ s
Set CGRAM Address	0	0	0	1	AC 5	AC 4	AC 3	AC 2	AC 1	AC 0	Set CGRAM address in address counter.	37 μ s
Set DDRAM Address	0	0	1	AC 6	AC 5	AC 4	AC 3	AC 2	AC 1	AC 0	Set DDRAM address in address counter.	37 μ s

Read Busy Flag and Address	0	1	B F	AC 6	AC 5	AC 4	AC 3	AC 2	AC 1	AC 0	Whether during internal operation or not can be known by reading BF. The contents of address counter can also be read.	0 μ s
Write Data to RAM	1	0	D 7	D6	D5	D4	D3	D2	D1	D0	Write data into internal RAM (DDRAM/CGRAM).	37 μ s
Read Data from RAM	1	1	D 7	D6	D5	D4	D3	D2	D1	D0	Read data from internal RAM (DDRAM/CGRAM).	37 μ s

Note:

Be sure the ST7066U is not in the busy state (BF=0) before sending an instruction from the MPU to the ST7066.

If an instruction is sent without checking the busy flag , the time between the first instruction and next instruction

Will take much longer than the instruction time itself. Refer to Instruction Table for the list of each instruction execution time .

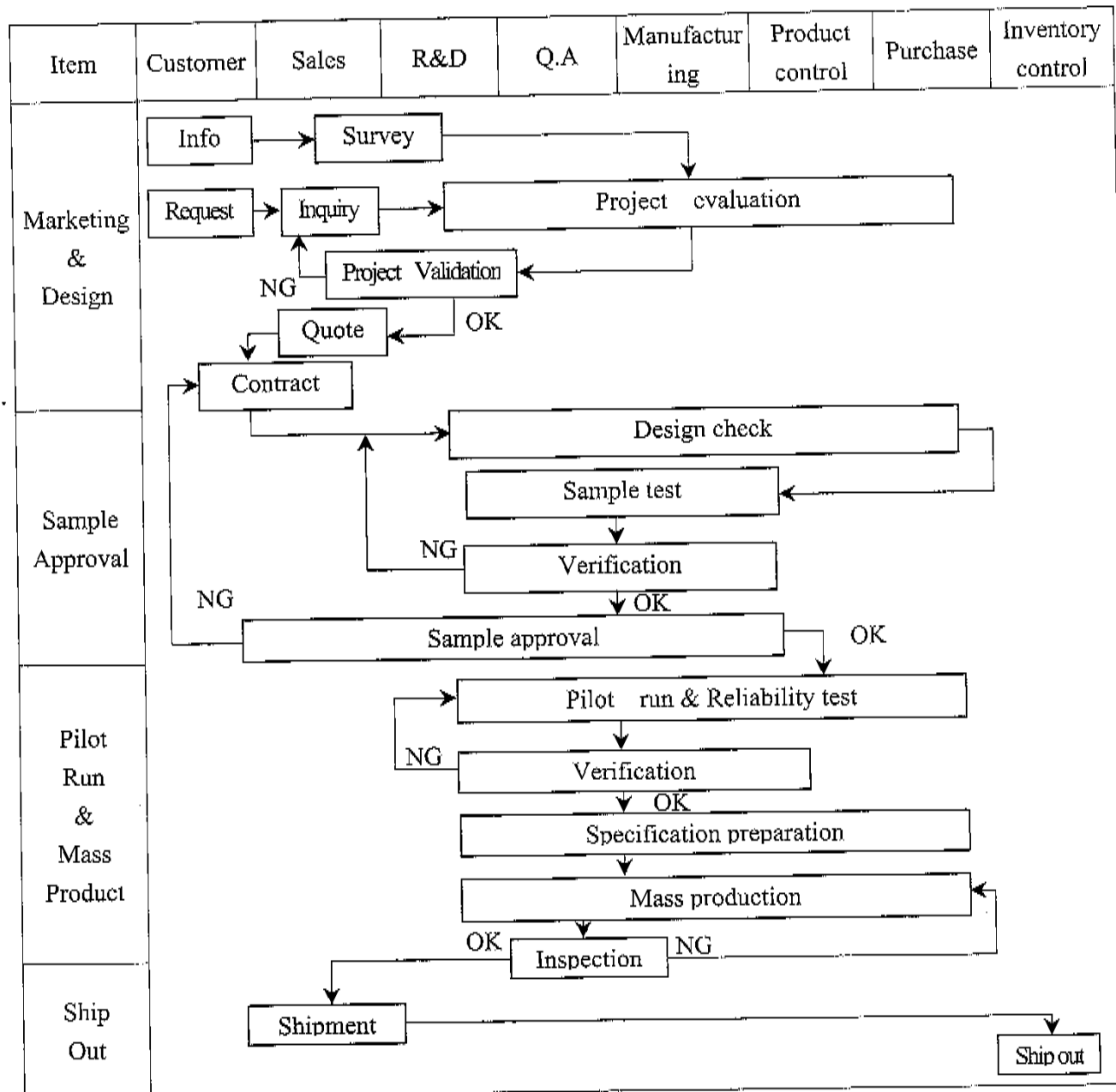
2.5 Character Pattern

CHARACTER PATTERN(SO/HO/FA,WA)

Pattern	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
CG xxxx0000 RAM (1)																
xxxx0001 (2)																
xxxx0010 (3)																
xxxx0011 (4)																
xxxx0100 (5)																
xxxx0101 (6)																
xxxx0110 (7)																
xxxx0111 (8)																
xxxx1000 (1)																
xxxx1001 (2)																
xxxx1010 (3)																
xxxx1011 (4)																
xxxx1100 (5)																
xxxx1101 (6)																
xxxx1110 (7)																
xxxx1111 (8)																

3. QUALITY ASSURANCE SYSTEM

3.1 Quality Assurance Flow Chart



Item	Customer	Sales	R&D	Q.A	Manufacturing	Product control	Purchase	Inventory control
Sales Service	<pre> graph TD Info[Info] --> Claim[Claim] Claim --> Failure[Failure analysis] Failure --> Corrective[Corrective action] Corrective --> Tracking[Tracking] Info --> Analysis[Analysis report] </pre>							
Q.A Activity	<div>1. ISO 9001 Maintenance Activities</div> <div>2. Process improvement proposal</div> <div>3. Equipment calibration</div> <div>4. Education And Training Activities</div> <div>5. Standardization Management</div>							

3.2 Inspection Specification

Inspection Standard : MIL-STD-105E Table Normal Inspection Single Sampling Level II .

Equipment : Gauge 、 MIL-STD 、 Powertip Tester 、 Sample .

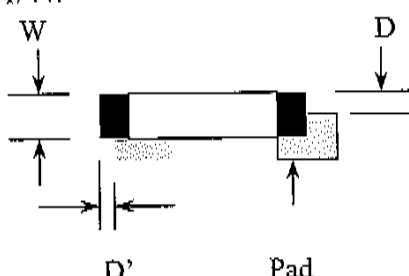
IQC Defect Level : Major Defect AQL 0.4; Minor Defect AQL 1.5 .

FQC Defect Level : 100% Inspection .

OUT Going Defect Level : Sampling .

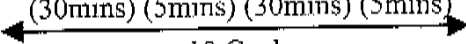
Specification :

NO	Item	Specification	Judge	Level
1	Part Number	The part number is inconsistent with work order of production	N.G.	Major
2	Quantity	The quantity is inconsistent with work order of production	N.G.	Major
3	Electronic characteristics of LCM $A = (L + W) \div 2$	The display lacks of some patterns.	N.G.	Major
		Missing line.	N.G.	Major
		The size of missing dot, A is $> 1/2$ Dot size	N.G.	Major
		There is no function.	N.G.	Major
		Output data is error	N.G.	Major
4	Appearance of LCD $A = (L + W) \div 2$	Material is different with work order of production	N.G.	Major
		LCD is assembled in inverse direction	N.G.	Major
		Bezel is assembled in inverse direction	N.G.	Major
		Shadow is within LCD viewing area + 0.5 mm	N.G.	Major
		The diameter of dirty particle, A is > 0.4 mm	N.G.	Minor
	Dirty particle (Including scratch 、 bubble)	Dirty particle length is > 3.0 mm, and 0.01 mm $<$ width ≤ 0.05 mm	N.G.	Minor
		Display is without protective film	N.G.	Minor
		Conductive rubber is over bezel 1mm	N.G.	Minor
		Polarizer exceeds over viewing area of LCD	N.G.	Minor
		Area of bubble in polarizer, A > 1.0 mm, the number of bubble is > 1 piece.	N.G.	Minor
5	Appearance of PCB $A = (L + W) \div 2$	0.4 mm $<$ Area of bubble in polarizer, A < 1.0 mm, the number of bubble is > 4 pieces.	N.G.	Minor
		Burned area or wrong part number is on PCB	N.G.	Major
		The symbol, character, and mark of PCB are unidentifiable.	N.G.	Minor
		The stripped solder mask , A is > 1.0 mm	N.G.	Minor
		0.3 mm $<$ stripped solder mask or visible circuit, A < 1.0 mm, and the number is ≥ 4 pieces	N.G.	Minor
		There is particle between the circuits in solder mask	N.G.	Minor
		The circuit is peeled off or cracked	N.G.	Minor
		There is any circuits risen or exposed.	N.G.	Minor
		0.2 mm $<$ Area of solder ball, A is ≤ 0.4 mm	N.G.	Minor
		The number of solder ball is ≥ 3 pieces	N.G.	Minor
		The magnitude of solder ball, A is > 0.4 mm.	N.G.	Minor

NO	Item	Specification	Judge	Level
6	Appearance of molding $A=(L+W)\div 2$	The shape of modeling is deformed by touching.	N.G.	Major
		Insufficient epoxy: Circuit or pad of IC is visible	N.G.	Minor
		Excessive epoxy: Diameter of modeling is $>20\text{mm}$ or height is $>2.5\text{mm}$	N.G.	Minor
		The diameter of pinhole in modeling, A is $>0.2\text{mm}$.	N.G.	Minor
7	Appearance of frame $A=(L+W)\div 2$	The folding angle of frame must be $>45^\circ +10^\circ$	N.G.	Minor
		The area of stripped electroplate in top-view of frame, A is $>1.0\text{mm}$.	N.G.	Minor
		Rust or crack is (Top view only)	N.G.	Minor
		The scratched width of frame is $>0.06\text{mm}$. (Top view only)	N.G.	Minor
8	Electrical characteristic of backlight $A=(L+W)\div 2$	The color of backlight is nonconforming	N.G.	Major
		Backlight can't work normally.	N.G.	Major
		The LED lamp can't work normally	N.G.	Major
		The unsoldering area of pin for backlight, A is $>1/2$ solder joint area.	N.G.	Minor
		The height of solder pin for backlight is $>2.0\text{mm}$	N.G.	Minor
10	Assembly parts $A=(L+W)\div 2$	The mark or polarity of component is unidentifiable.	N.G.	Minor
		The height between bottom of component and surface of the PCB is floating $>0.7\text{mm}$	N.G.	Minor
		$D > 1/4W$ 	N.G.	Minor
		End solder joint width, D' is $>50\%$ width of component termination or width of pad	N.G.	Minor
		Side overhang, D is $>25\%$ width of component termination.	N.G.	Minor
		Component is cracked, deformed, and burned, etc.	N.G.	Minor
		The polarity of component is placed in inverse direction.	N.G.	Minor
		Maximum fillet height of solder extends onto the component body or minimum fillet height is $<0.5\text{mm}$.	N.G.	Minor

4. RELIABILITY TEST

4.1 Reliability Test Condition

NO	Item	Test Condition	
1	High Temperature Storage	Storage at $80 \pm 2^{\circ}\text{C}$ 96~100 hrs Surrounding temperature, then storage at normal condition 4hrs	
2	Low Temperature Storage	Storage at $-30 \pm 2^{\circ}\text{C}$ 96~100 hrs Surrounding temperature, then storage at normal condition 4hrs	
3	High Temperature /Humidity Storage	1.Storage 96~100 hrs $60 \pm 2^{\circ}\text{C}$, 90~95%RH surrounding temperature, then storage at normal condition 4hrs. (Excluding the polarizer). or 2.Storage 96~100 hrs $40 \pm 2^{\circ}\text{C}$, 90~95%RH surrounding temperature, then storage at normal condition 4 hrs.	
4	Temperature Cycling	$-20^{\circ}\text{C} \rightarrow 25^{\circ}\text{C} \rightarrow 70^{\circ}\text{C} \rightarrow 25^{\circ}\text{C}$ $(30\text{mins}) (5\text{mins}) (30\text{mins}) (5\text{mins})$ <div style="text-align: center;">  10 Cycle </div>	
5	Vibration	10~55Hz (1 minute) 1.5mm X,Y and Z direction * (each 2hrs)	
6	ESD Test	Air Discharge: Apply 6 KV with 5 times discharge for each polarity +/-	Contact Discharge: Apply 250V with 5 times discharge for each polarity +/-
		Testing location: Around the face of LCD	Testing location: 1.Apply to bezel. 2.Apply to Vdd, Vss.
7	Drop Test	Packing Weight (Kg)	Drop Height (cm)
		0 ~ 45.4	122
		45.4 ~ 90.8	76
		90.8 ~ 454	61
		Over 454	46

5. PRECAUTION RELATING PRODUCT HANDLING

5.1 SAFETY

- 5.1.1 If the LCD panel breaks , be careful not to get the liquid crystal to touch your skin.
- 5.1.2 If the liquid crystal touches your skin or clothes , please wash it off immediately by using soap and water.

5.2 HANDLING

- 5.2.1 Avoid any strong mechanical shock which can break the glass.
- 5.2.2 Avoid static electricity which can damage the CMOS LSI—When working with the module , be sure to ground your body and any electrical equipment you may be using.
- 5.2.3 Do not remove the panel or frame from the module.
- 5.2.4 The polarizing plate of the display is very fragile. So , please handle it very carefully ,do not touch , push or rub the exposed polarizing with anything harder than an HB pencil lead (glass , tweezers , etc.)
- 5.2.5 Do not wipe the polarizing plate with a dry cloth , as it may easily scratch the surface of plate.
- 5.2.6 Do not touch the display area with bare hands , this will stain the display area.
- 5.2.7 Do not use ketonics solvent & aromatic solvent. Use with a soft cloth soaked with a cleaning naphtha solvent.
- 5.2.8 To control temperature and time of soldering is $280 \pm 10^{\circ}\text{C}$ and 3-5 sec.
- 5.2.9 To avoid liquid (include organic solvent) stained on LCM .

5.3 STORAGE

- 5.3.1 Store the panel or module in a dark place where the temperature is $25^{\circ}\text{C} \pm 5^{\circ}\text{C}$ and the humidity is below 65% RH.
- 5.3.2 Do not place the module near organics solvents or corrosive gases.
- 5.3.3 Do not crush , shake , or jolt the module.

5.4 TERMS OF WARRANTY

- 5.4.1 Applicable warrant period
The period is within thirteen months since the date of shipping out under normal using and storage conditions.
- 5.4.2 Unaccepted responsibility
This product has been manufactured to your company's specification as a part for use in your company's general electronic products. It is guaranteed to perform according to delivery specifications. For any other use apart from general electronic equipment , we cannot take responsibility if the product is used in nuclear power control equipment , aerospace equipment , fire and security systems or any other applications in which there is a direct risk to human life and where extremely high levels of reliability are required.