Noritake CU20026SCPB-T23A

2 Line x 20 Character VFD display module

Unofficial device datasheet V1.0

Monday, December 08, 2014

Table of Contents

2 x 20 Blue VFD Display Module

- 2 Line x 20 Character 5x7 Dot Matrix VFD Display module used in Nortel Millennium Pay Phones.
- This display device is similar to many such VFD's using the Noritake or NEC VFD protocols.
- The module was designed in the early 90's or so. The phones were produced in the latter half of the 90's to the first few years of this century.
- Note that the phones spent a lot of time displaying "`Please lift receiver`" on line 1, so that is probably badly burnt into the display if the telephone was ever in service.
- There are many implementations of those protocols amongst the various similar devices for which data is available. Each had its own idiosyncrasies and features on top of the base functions that vary by device.
- This device appears to be a custom featured device built for Nortel for use with these telephones. Noritakes's web site indicates that custom modules may be created for as few as 1000 unit per year quantities, so customized modules are not uncommon. They provide a wide range of options that might be selected for a module.
- This document and the associated sample code provide pretty much full documentation for the Nortel variant of this display. Any additional data would be appreciated.
- Anyone who knows of other features the device supports or errors or omissions in this documentation or sample code is welcome to contact the author at: ftl at functech dot com.

Introduction

Credits

- Many thanks to a group who got the base functions of the display module working and provided some sample code by analyzing logic traces in a working application.
- Thanks to another source that provided some documentation on a similar module.
- The datasheet for the Noritake CU40026SCPB-S20A describes a module extremely similar to this one and provided some helpful guidance even though it is a 2x80 character module. A copy of that datasheet is included in the appendix.
- The rest of this comes from the author's analysis of datasheets from similar modules, analysis of the module's use in an application and much experimentation with code and tracing logic lines.

Notes on Nomenclature:

- All bits within a byte are numbered with D0 being the least significant (LSB) and D7 being the most significant (MSB). This is consistent with Arduino standard and most datasheets.
- Line or bit names with a leading under score ('_') imply an active low line, this is normally shown in datasheets with a bar overtop to indicate that the low value provides the function and a high value is the idle or do-nothing state. This format is used through this document and in the provided sample code.

- Any data shown in hexadecimal notation is displayed using the C syntax of “0x00”. This is consistent with usage in the sample C code.
- All logic levels are assumed to be 5V based. No testing was done to determine actual logic switching levels.
- An Arduino Uno with an ATmega168 running at 16Mhz with 5V logic levels was used as the Host for all testing. The Arduino C code would generally be easy to port to most other platforms if required.

Copyright

Disclaimer

Physical Interface

Electrical Specifications

- The device runs on 5V and appears to be compatible with standard 5V logic levels. No hard data is available on the exact limits of the electrical interface.
- The specifications listed in the datasheet for the Noritake CU40026SCPB-S20A (included in the appendix) are probably appropriate as that device is the most similar one I found, and appears to be from the same era.
- Testing indicates current consumption varies from 250ma to 370ms.
    - 250ma Minimum brightness with normal characters displayed.
    - 320ma Full brightness with normal characters displayed.
    - 370ma Full brightness with all pixels lit.

Pin Out

The VFD is provided with a 26 pin IDC connector on the end of a 10 inch 26 conductor ribbon cable. The pin out looking at the female side of the connector at the end of the cable is as shown below. This is the

same configuration as when looking at the solder side of the circuit board. This is a custom design created for the original user of the display and is not similar to any other device I've found.

						xKEY						
1	3	5	7	9	11	13	15	17	19	21	23	25
D7	D6	D5	D4	D3	D2	D1	D0	_WR	X2	X3	_TS	RS
2	4	6	8	10	12	14	16	18	20	22	24	26
+5V	+5V	+5V	+5V	+5V	Gnd	Gnd	Gnd	Gnd	RS	Gnd	Gnd	Gnd

Lines	Comments
+5V	All +5V lines are connected together at the board. Current draw is about 300ma. It is sufficient to connect to any one of these. Using them all may produce better noise immunity in the cable.
Gnd	All ground pins are connected together at the board. It is sufficient to connect to any one of these. Using them all may produce better noise immunity in the cable.
D0 – D7	Data bus with the normal sorts of 5V logic levels. D0 is LSB, D7 is MSB Normally an input to the display and output from the uP. Some documentation indicates that using the read function, D0 and D1 can provide status information to the uP and D2-D7 can be used to read cursor position. The read functions do change the line levels, but no data appears to be transferred. The Dx lines appear to be Host → Device only. No information is available for 3.3V or lower voltage applications.
_WR	Write – Normally high. Low to modify data lines. Transition high to latch the data and execute the command on the VFD.
AD	Address / Data – Specifies Data or Command mode. 0-Data, 1-Cmd
_RD	Specifies Read mode. 1-Read, 0-Normal. Although defined, could not find any condition in which data is returned.
_CS	Chip Select. 0 – Selected. 1 – High-Z interface. When the device is not selected, D0-D7 are in a High-Z state. There is no data on how the other lines are affected by _CS. _CS was not really tested, but in one examined application, D0-D7 was shared with several other I/O devices with no issues by making use of the _CS capabilities.
_TS	Test Mode - Hold low for 150ms during HW reset for test mode display. Probably not hi-Z when _CS=1.
RS	Hardware reset – Active high. Pulse high to reset chip. Pulsing RS is the only way other than removing power to cancel test mode. Probably not hi-Z when _CS=1.

Minimum Requirements for Physical Interface

- Although 14 lines are defined that can be used to interface with the display (in addition to power and ground), in many applications they are not all required to be connected to pins on the controlling microcontroller.
- The display can be used with as few as 8 lines, while 9 or 10 would be the most likely configurations. The following are some examples of possibilities and the limitations they impose.
- The sample code assumes a uP connection to all 14 lines to demonstrate all chip functions.

8-Lines

- Use only D0-D6 and _WR. Others are tied high or low as follows:

    - D7 Low – High order data bit will always be zero.
    - _CS Low – Chip is always selected
    - _AD Low – Only data commands are accepted
    - TS High
    - RS Low
- Display character codes from 0-127 may be used, including the control codes. If you are not using any of the built-in special characters, you won't miss them.
- The biggest issue will be the limited ability to move the cursor. After reset, the cursor will be in the first position. If you always write exactly 40 characters with each I/O sequence, the cursor should remain in positon zero after each sequence.
- There is no ability to move the cursor back to the beginning of a line if you have not kept track of what you are sending.

Additional Lines to Enable Functionality

- Connect the AD line to a uP pin:
    - You will be able to use the cursor positioning command and the software reset command.
    - Cursor positioning is very useful, since it is the only way to get back to putting the cursor in position 0 of the display from any mode you may be in.
    - Cursor positioning is also the easiest way to move a visible cursor on the display if user input is occurring. It could probably be done with some very careful coding, but it would be problematic.
- Connect D7 line to a uP pin:
    - You will be able to use the additional built-in characters in positions 128-255.
    - These characters are useful for non-English languages.
    - Without D7, custom character bit maps may still be used as long as they are stored below 128 (0x80) in the character map. See the various notes on custom characters to see if they can do the trick for you. You can only define two custom bitmaps at a time.
- Connect _CS to a uP pin:
    - The device supports a chip-select function. If _CS is high, the device pins will go into a high-impedance state, so the uP pins they connect to can be shared with other devices as long as they support a chip-select functio0n as well.
    - D0 – D7 certainly support the chip select function.
    - I did not test if the other control lines do as well. Obviously the _CS line itself cannot be shared.
- The hardware reset (RS) and test (T0) lines would rarely be required in an application, so they can usually be set to fixed values and not be missed.
    - A software initiated reset is available via the Cmd/Cursor positioning interface that can be used instead of hardware reset if the AD line is available.
    - If a hardware test is initiated via the T0 line, the only way out is to power-cycle the display or do a hardware reset. A software reset will not terminate the test process.

Device Speed

- After issuing a command to the device, the device needs a certain amount of time to process that command before it can accept another commands.
- Issuing another command (lowering then raising _WR) too soon, can cause that command to be ignored, or can be undefined and cause strange results on the display.
- In most cases commands issued too soon just appeared to be ignored.
- Most commands will complete in much less than 50us, often 20us, so characters can generally be written to the display at a rate of 20,000 per second or more.
- At times though commands will take a bit longer. It may be time to do a vertical scroll, or the command is being issued by the host just as a display refresh begins. The documented guaranteed delays are more in the 100us per command range.
- The most significant command delay is with Clear (FF or ox0C). that is documented as taking 560us. Resets are also much slower (150ms).

Monitoring Device Ready Status

- Although some documentation indicates that the device maintains a status bit that can be queried, the tested device did not ever return a busy status immediately after receiving what should have been a long duration command.
- The good news is that the controller is much faster than the usual HD44780 type chips usually used to control LCD's where waiting on a status becomes a necessity.

Using Fixed Delays

- The only method of successfully interfacing with the device involves waiting a fixed time period after each command for the command to complete before executing the next.
- Although this works well, it is slower that monitoring the device status, and unless the delays are set conservatively, can cause overrun errors in some cases. For instance writing a character is usually a relatively fast operation, but if writing that character causes the display to scroll, it will take longer.
- A table of known required delays exists in the sample code. That data comes from testing and analysis of the display's use in a working application.
- For all commands except clearing the screen and resetting the device, about 100us seems to be sufficient time for the display to complete a command.
- The sample code includes a simple scheme where the processor does not have to wait for the display immediately after issuing a command, but only waits if it has not been long enough for the next command when that command is issued.

Logical interface

- Various commands can be sent to the device via its logical protocol.

- No command makes a non-volatile change to the device configuration. The display will completely reset to its default state after a reset command (hardware or software) or a power off cycle.
- The operation to be performed is first influenced by the state of the control lines:

_CS	AD	_WR	Command State
0	0	0→1	Data write – sends a byte to the display.
0	1	0→ 1	Command write – cursor position or SWreset.
1	x	x	No operation

Command Protocol

- There are two types of commands that may be sent to the display: Data, Command.
- Data is used for most operations on the display.
- Cmd is used to position the cursor to a specific location or reset the chip via software.

Character Positions

- The individual character positons can be addressed with the cursor-move command.
- They are also addressed in this fashion for setting blinking mode on and off. Shaded positions can be set to blink. The blinking rate cannot be changed.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	**19**
20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	**39**

Character Display ROM Definitions

- The display has a built-in character generator that will display suitable characters.
- The character ROM is customized for the original purpose of the display, but is still useful for many purposes.
- The ROM codes for the display that was tested are shown below. There may have been other customized version of this display module used over time, so you r results may differ slightly.
- The characters from 0x20 through 0x7F are normal ASCII.
- The following table shows the ranges of characters that are available in the 0-255 address space.
- Codes not listed are null. That is, they do not change any characters on the display, nor do they change the mode in any way. The cursor is not moved either.

Code Range	Type	Description
0x0A	Control	Line Feed (LF) – Moves the cursor position to the same column in the other row. Scrolls the display up one line in Vertical scroll mode if it was already on the second row.
0x0C	Control	Form Feed (FF) – Clears the display. The cursor position is not moved.
0x11	Control	DC1 – Sets the display into normal display mode. Displayable characters are written at the current cursor position and the cursor is moved up one position. The Cursor wraps from position 39 to position 0.
0x12	Control	DC2 – Sets the display into vertical scroll mode. Whenever the cursor wraps off the end of line 2, the characters in line 2 are copied to line 1 and the cursor is placed at the beginning of line 2.
0x13	Control	DC3 – Turns on the underline cursor in the current cursor position. The underline moves as the cursor moves.
0x14	Control	DC4 – Turns the cursor off.
0x15	Control	DC5 – Sets the display to horizontal scrolling mode. When the cursor moves off of the end of a line, the characters in that line are moved over one position to the left. The leftmost character is lost. The cursor is then placed on the last column of the line and the new character is written there. Ether line can be scrolled while the other is not touched.
0x16	Control	DC6 – Sets the display to blinking mode. This is a two-byte command. The second byte indicates which position should blink. Only positions 0, 19, 20,and 39 can blink. Blink commands to the other positions are ignored. Issue this for each position you want to blink.
0x17	Control	DC7 –Turns off blinking mode. All blinking positions will be reset.
0x1A	Control	SUB –Sets the display brightness. This is a two byte command with the second byte selecting the brightness level. Supported levels are: 0, 16, 32, 48, with 0 being the brightest and 48 the dimmest..
0x1B	Control	ESC – Create a user defined character bitmap. This is a 7-byte command: ESC, <Char to define>, D0, D1, D2, D3, D4 More details are described below.
0x20	Display	Space (‘ ‘)
0x21 – 0x2F	Display	ASCII special characters ! ″ # $ % & ′ ( ) * + ′ - . /
0x30 – 0x39	Display	ASCII numbers 0 1 2 3 4 5 6 7 8 9
0x3A – 0x40	Display	ASCII special characters ; < = > ? @
0x41 – 0x5A	Display	ASCII capital letters: A – Z
0x5B – 0x60	Display	ASCII special characters ; [\ ] ^ _ `
0x61 – 0x7A	Display	ASCII lower case letters: a – z

Code Range	Type	Description
0x7B – 0x7E	Display	ASCII special characters { | } ~
0x7F – 0xA0	Null	Appears to do nothing, and displays nothing. Does not move the cursor position when sent to the display.
0xA1 – 0xB3	Accents	Vowels with grave, acute, circumflex, umlaut, and cedilla accents.
0xB4 – 0xC0	Blank	Display as blanks. Some display may have some special symbols in 0xB4 – 0xB8.
0xC0 – 0XFF	Japanese	Japanese Katana character set.

Custom Character Bitmaps

- Custom character bitmaps may be defined for any character in the ROM array, including control characters. It will probably even work for the characters that by default do not display anything.
- If you change a control character, it becomes a display character until reset, or it is freed by defining more custom characters.
- You can only define two custom characters. Once you define a third, the oldest customer character you defined will be removed, and revert back to its default, including control functions.
- Once a custom bitmap is displayed, if the bitmap for that character is changed, all occurrences of the customer character will change.
- The display's ability to store custom bitmap is very limited. It only has two bitmap buffers that are used on a rotating basis. Whenever a buffer is used, it replaces the oldest character that is defined.
- Because you do not have direct control of which buffer is used for which character, it is difficulty to dynamically load bit maps and keep track of which bitmap will be used.
- Basically you should limit the use to loading up to two bitmaps during initialization and then use those characters.
- A bitmap change will always use the next buffer, even if it is re-defining a character that already has a bit map assigned. The implications are:
    - Redefining a single character will erase the definition for another character.
    - There is a bug in this process: If you redefine an overridden character, it does not update the character to use the new buffer so you don't see the update.
- If you define two characters or re-define the same character twice it works ok.
- A custom bitmap is loaded using the ESC (0x1B) control code. When you send this command, the next 6 bytes sent to the display will be used to create the custom bitmap.
- The command sequence is as follows:
    - 0x1B – ESC Start the process
    - Character code to be redefined. This can be any code, including 0x1B, that will disable the ability to load another code until a hardware rest or power cycle is performed.
    - Five bytes of data to be used for the bitmap. The bit arrangement in these bytes is a bit strange, and is detailed in the following tables.

Data Sent to display:

Bit# →	MSB 7	6	5	4	3	2	1	LSB 0
ESC – 0x1B	0	0	0	1	1	0	1	1
Char-Code	x	x	x	x	x	x	x	x
Byte #1	H	G	F	E	D	C	B	A
Byte #2	P	O	N	M	L	K	J	I
Byte #3	X	W	V	U	T	S	R	Q
Byte #4	6	5	4	3	2	1	Z	Y
Byte #5	.	.	.	.	a	9	8	7

Pixels Displayed for Character:

The bits in the data bytes above are displayed in the following pixel positions. (0=Off, 1=On)

A	**C**	**E**	**G**	**I**
K	**M**	**O**	**Q**	**S**
U	**W**	**Y**	**1**	**3**
5	**7**	**9**	**B**	**D**
F	**H**	**J**	**L**	**N**
R	**T**	**V**	**X**	**Z**
2	**4**	**6**	**8**	**a**

P	**P**	**P**	**P**	**P**

Appendix-1 Sample Code

The following sample code demonstrates many of the features of the display. It was developed using the Arduino V1.06 IDE and was downloaded to and tested on an Arduino UNO with an ATMega168. The code documentation provides additional; interface details.

Hopefully the source code file will not get too separated from this document, but just in case here is the sample source code:

```
//CPAGE=============================================================================
//
//  Demo for Nortel / Noritake 20x2 VFD Display
// ==========================================
//
// Copyright:
// ----------
//  Public domain
//
// Version History:
// ----------------
//  V1.0 - 2014-12-05 - Initial code.
//                    - Development Environment: Arduino V1.06
//
// Overview:
// ---------
// Run a Noritake VFD module from a Nortel Millenium pay phone through many of its paces.
// - A base to develop a fully functional driver.
// - Testing environment to discover features of the module.
//
// Hardware:
// ---------
//   CPU board: Solarbotics Freeduino V2.1 (old version)
//   - Atmel ATmega128 @ 16MHz, 16K flash, 1K RAM, 512 bytes EEPROM.
//   Configure as: "Arduino Diecimala or Dumilanove w/ATmega168"
//   Serial monitor: 115200
//
//==================================================================================
#include <stdlib.h>

//CPAGE=============================================================================================
//  Pin definitions
//
// - The following pins must be connected to the VFD as follows in the hardware setup.
// - Any Arduino digital capabile pin may be used for any function.
// - All port manipulation is performed by high-level Arduino functions.
//===================================================================================================
//       Software  Digital    Hardware             Display  Display
//         Name     Pin#       Pin-ID      Mode     Pin#    Pin-Description
#define   VFD_D7     2     //    D2        Out       15       D0    - Data bus
#define   VFD_D6     3     //    D3        Out       13       D1
#define   VFD_D5     4     //    D4        Out       11       D2
#define   VFD_D4     5     //    D5        Out        9       D3
#define   VFD_D3     6     //    D6        Out        7       D4
#define   VFD_D2     7     //    D7        Out        5       D5
#define   VFD_D1     8     //    D8        Out        3       D6
#define   VFD_D0     9     //    D9        Out        1       D7    - Data bus
#define   VFD__WR   14     //    D14-A0    Out       17       _WR   - Write
#define   VFD_AD    15     //    D15-A1    Out       19       AD    - Address/Data mode
#define   VFD__RD   16     //    D16-A2    Out       21       ST    - Read mode
#define   VFD__CS   17     //    D17-A3    Out       23       _CS   - Chip select
#define   VFD__TS   18     //    D18-A4    Out       25       _TS   - Test mode
#define   VFD_RS    19     //    D19-A5    Out       20       RS    - Hardware reset

//CPAGE=============================================================================================
//  VFD Control Constants
//===================================================================================================
```

```
// Display parameters and cursor positioning constants used in command mode
#define VFD_NumLines                2    // 2 lines
#define VFD_LineLength             20    // 20 characters per line
#define VFD_Addr_L1_First           0    // Cursor position - Top-left
#define VFD_Addr_L2_First          20    // Cursor position - Bot-Left
#define VFD_Addr_L1_Last           19    // Cursor position - Top-Right
#define VFD_Addr_L2_Last           39    // Cursor position - Bot-Right

#define VFD_Cmd_SWReset          0x40    // Software reset - from cmd mode only
#define VFD_Cmd_Read_Pos         0x41    // Read cursor posn - this may not work

// Control codes that can be sent to the display in data mode
// - See documentation for more details.
#define VFD_Ctl_LF               0x0A    // Move cursor down one line
#define VFD_Ctl_FF               0x0C    // Clear display, does not move cursor

#define VFD_Mode_Normal          0x11    // DC1 Normal display mode
#define VFD_Mode_VertScroll      0x12    // DC2 Sroll vertically
#define VFD_Mode_Cursor_On       0x13    // DC3 Turn cursor on
#define VFD_Mode_Cursor_Off      0x14    // DC4 Turn cursor off
#define VFD_Mode_HorzScroll      0x15    // DC5 Scroll horizontially
#define VFD_Mode_Blinking_On     0x16    // DC6 Blink a corner positions (2-byte code)
#define VFD_Mode_Blinking_Off    0x17    // DC7 Turn off all blinking

#define VFD_Set_Brightness       0x1A    // set intensity level (2-byte code)
#define VFD_Set_Char_Bitmap      0x1B    // ESC - Define bitmap (7-byte code)

// Time in microseconds for the display to process variopus types of command.
// - These are very conservative top allow for worst-case situations. Thay come from
//   analysis of the display being used in an application.
// - Do you own testing to determine smaller values for your application
//   in specific write sequences if speed matters that much to you.
// - Many functions worked fine down to 10-20us delays between commands.
// - It would be better to monitor a busy flag, but it does not seem that the
//   display provides that status information.
// - The display is much faster than HD44780 type LCD displays that need to support a busy flag.
// - This sample code uses a simplistic set of delays.
//   - Cmds are either a SWReset or a Cursor_Posn
//   - Data is either VFD_Ctl_FF (clear) or something else for which VFD_Delay_Mode_Normal will
do.
#define VFD_Hold_Test          150000    // Hold test line low for 150ms to initiate.
#define VFD_Delay_Test         100000    // Allow 100ms for test to initiate.
#define VFD_Delay_HWReset      150000    // Allow 150ms for HW resets to complete.
#define VFD_Delay_SWReset      100000    // Allow 100ms for SW resets to complete.
#define VFD_Delay_Reset_Pulse       5    // Hold reset line for this long to reset.
#define VFD_Delay_Cursor_Posn      40    // Move the cursor position
#define VFD_Delay_Clear           560    // Clear the display
#define VFD_Delay_DC1             100    // Change mode to normal mode
#define VFD_Delay_Mode_Normal     120    // Write a character in normal mode
#define VFD_Delay_Cursor_Mode     100    // Change cursor mode: DC3, DC4, DC5, DC6, DC7
#define VFD_Delay_Set_Blink_Posn   70    // Set a position to blinking mode
#define VFD_Delay_Mode_HorizScroll 90    // Write a character in horizontal scrolling mode

// Constants used to set the brightness of the display.
#define VFD_Brightness_100       0x00    // 100%
#define VFD_Brightness_75        0x10    //  75%
#define VFD_Brightness_50        0x20    //  50%
#define VFD_Brightness_25        0x30    //  25%

//CPAGE=============================================================================================
// Global Data
//==================================================================================================
unsigned long     LastCmdTime=0;     // micros() timer value when last command was executed.
unsigned long     VFDCmdWait=500;    // us to wait until the next command may be issued.
char              VFD_Buffer[40];    // Full display buffer for VFD.

//CPAGE=============================================================================================
// setup
// - Executed once after device reset.
//==================================================================================================
void setup() {
```

```
  Serial.begin(115200);              // Make sure PC virtual serial port is set to this.
  Serial.setTimeout(1);              // Don't wait for stuff!
  Serial.println("Starting . . . .");

  // Set up pins for VFD communications
  pinMode(VFD_D0,  OUTPUT);
  pinMode(VFD_D1,  OUTPUT);
  pinMode(VFD_D2,  OUTPUT);
  pinMode(VFD_D3,  OUTPUT);
  pinMode(VFD_D4,  OUTPUT);
  pinMode(VFD_D5,  OUTPUT);
  pinMode(VFD_D6,  OUTPUT);
  pinMode(VFD_D7,  OUTPUT);
  pinMode(VFD__WR, OUTPUT);
  pinMode(VFD_AD,  OUTPUT);
  pinMode(VFD__RD, OUTPUT);
  pinMode(VFD__CS, OUTPUT);
  pinMode(VFD__TS, OUTPUT);
  pinMode(VFD_RS,  OUTPUT);

  // Start the initial states of the control lines.
  // - The data lines don't matter at this point.
  digitalWrite(VFD__WR, HIGH);   // _WR - Write mode - Active low)
  digitalWrite(VFD_AD,  LOW);    // AD  - Address/Data mode - Always data?
  digitalWrite(VFD__RD, HIGH);   // RD  - Read mode: 0-Data/Status Read 1-Data/Cmd Write
  digitalWrite(VFD__CS, LOW);    // _CS - Chip select - Always selected?
  digitalWrite(VFD__TS, HIGH);   // _TS - Test mode bit - active low
  digitalWrite(VFD_RS,  LOW);    // RS  - hardware reset line

  LastCmdTime=0;                // We have not issued a command yet, so no need to wait.
  VFD_HW_Reset();               // Do a hardware reset just because we can!

} //  Setup

//CPAGE=========================================================================================
// loop
// - executed in an endless loop until the device is reset or powered down.
//===============================================================================================
void loop() {

  //CPAGE---------------------------------------------------------------------------------------
  //   Test mode and resets
  //---------------------------------------------------------------------------------------------

  // Put the display into test mode for while. The only way to cancel it is with a device
hardware reset.
  // - A software reset will not reset test mode.
  VFD_Test_Mode();                              // Go into test mode
  delay(5000);                                  // Wait to see some of the display

  // Now do a hardware reset to clear test mode. It is the only way out of test.
  VFD_HW_Reset();

  //CPAGE---------------------------------------------------------------------------------------
  //   Display updates via full screen buffer
  // - With the cuirsor on you can see it moving as the display is updated.
  // - The display looks much better with cursor off.
  // - It takes about 7ms to do all of the work in the loop, so although the counter decrements
  //   by 25, we only wait 18ms at the end.
  //---------------------------------------------------------------------------------------------
  char Num_Buff[6];

  // User full screen updates to dislay a fast moving count down.
  memset(VFD_Buffer, ' ', sizeof(VFD_Buffer));
  VFD_Str_To_Buffer(VFD_Buffer,    "Buffer demo:");
  VFD_Str_To_Buffer(VFD_Buffer+20 ,"xxxxx ms to go!");
  VFD_Send_Char(VFD_Mode_Cursor_Off);
  VFD_Send_Cmd(20);
  for (int i = 5000; i > 0; i-=25) {
    memset(VFD_Buffer + 20, ' ', 5);            // Blanks where the number goes
    itoa(i,Num_Buff,10);
```

```
  VFD_Str_To_Buffer(VFD_Buffer+20+(5-strlen(Num_Buff)),Num_Buff);
  VFD_Send_Full_Buffer(VFD_Buffer, 0);
  delay(18);                                  // Delay less than decrement to keep on time!
};

//CPAGE-------------------------------------------------------------------------------------
//   Blinking mode on corner characters.
// - With the cursor off.
//-------------------------------------------------------------------------------------------

memset(VFD_Buffer,' ', sizeof(VFD_Buffer));
VFD_Str_To_Buffer(VFD_Buffer,      "Blinking Stuff");
VFD_Str_To_Buffer(VFD_Buffer+25,        "LOOK!");
VFD_Send_Char(VFD_Mode_Cursor_Off);
VFD_Send_Full_Buffer(VFD_Buffer, 0);

// Insert some characters on the ends.
VFD_Send_Cmd(20);                             // Cursor to R2C1
VFD_Send_Char('X');
VFD_Send_Cmd(39);                             // Cursor to R2C20
VFD_Send_Char('X');

delay(500);

// Blink first and last columns of the 2nd line.
VFD_Send_Char(VFD_Mode_Blinking_On);           // Blink R2C1
VFD_Send_Char(20);                            // - Posn to blink
VFD_Send_Char(VFD_Mode_Blinking_On);           // Blink R2C20
VFD_Send_Char(39);                            // - Posn to blink

delay(2000);                                  // Gaze in wonder!

// Turn off blinking mode. This cancels all blinking positions.
VFD_Send_Char(VFD_Mode_Blinking_Off);

delay(1000);

//CPAGE-------------------------------------------------------------------------------------
//   Move the cursor around a bunch with control characters and commands.
//-------------------------------------------------------------------------------------------
memset(VFD_Buffer, ' ', sizeof(VFD_Buffer));
VFD_Str_To_Buffer(VFD_Buffer, "Cursor Moves:");
VFD_Str_To_Buffer(VFD_Buffer + 20, "01234567890123456789");
VFD_Send_Full_Buffer(VFD_Buffer,30);
VFD_Send_Char(VFD_Mode_Cursor_On);
delay(250);

// Line Feed - move to the next line or back to the top.
VFD_Str_To_Buffer(VFD_Buffer + 14, "LF");
VFD_Send_Full_Buffer(VFD_Buffer,30);
for (byte i = 0; i < 5; i++) {
  VFD_Send_Char(VFD_Ctl_LF);
  delay(250);
};

// Use the cursor move command to move the cursor around a bit just for fun..
VFD_Str_To_Buffer(VFD_Buffer + 14, "MV");
VFD_Send_Full_Buffer(VFD_Buffer,30);
for (byte i=0; i < 20; i++) {
  VFD_Send_Cmd(bitRead(i,0)*20 + i);
  delay(250);
};
delay(500);

// Clear it and see that the cursor is not moved by a clear!
VFD_Send_Cmd(10);
VFD_Send_Char(VFD_Ctl_FF);
VFD_Send_Str("Cursor not moved by FF!");
delay(2000);

//CPAGE-------------------------------------------------------------------------------------
```

```
//   Horizontal scrolling.
//-------------------------------------------------------------------------------------------
const char LongMsg[] = "This is a demonstration of builtin scrolling on the VFD. Blanks are "
                       "sent at the end to clear the message.";

memset(VFD_Buffer, ' ', sizeof(VFD_Buffer));
VFD_Str_To_Buffer(VFD_Buffer, "Horizontal Scrolling");
VFD_Send_Full_Buffer(VFD_Buffer, 0);
VFD_Send_Cmd(20);
VFD_Send_Char(VFD_Mode_HorzScroll);
for (const char * pCh = LongMsg; *pCh != 0; *pCh++) {
  VFD_Send_Char(*pCh);
  delay(100);
};

// Scroll the message off with blanks.
for (byte i = 0; i < 20; i++) {
  VFD_Send_Char(' ');
  delay(100);
};
VFD_Send_Char(VFD_Mode_Normal);

//CPAGE--------------------------------------------------------------------------------------
//   Vertical Scrolling.
//-------------------------------------------------------------------------------------------
memset(VFD_Buffer, ' ', sizeof(VFD_Buffer));
VFD_Str_To_Buffer(VFD_Buffer, "Vertical Scrolling");
VFD_Send_Full_Buffer(VFD_Buffer, 0);
delay(1000);
VFD_Send_Char(VFD_Mode_VertScroll);
for (byte i = 0; LongMsg[i] != 0; i++) {
  VFD_Send_Char(LongMsg[i]);
  delay(50);
};

// Scroll the message off with blanks.
for (byte i = 0; i < 40; i++) {
  VFD_Send_Char(' ');
  delay(50);
};
VFD_Send_Char(VFD_Mode_Normal);

//CPAGE--------------------------------------------------------------------------------------
//   Software Reset
//-------------------------------------------------------------------------------------------
VFD_Send_Cmd(VFD_Cmd_SWReset);                // Software reset
delay(1000);                                 // S/B a blank display

//CPAGE--------------------------------------------------------------------------------------
//   Show all of the printable characters and their codes.
//-------------------------------------------------------------------------------------------
byte Len;

VFD_Send_Char(0x0C);                           // Clear the display
VFD_Send_Cmd(0);
for (int Ascii=32; Ascii <= 255; Ascii++) {
  Len=sprintf(VFD_Buffer,"Ascii: %02X %3d",Ascii,Ascii);

  // Put our message on line 1
  VFD_Send_Cmd(0);        // Force it to position 0
  for (byte i=0;  i < Len; i++) VFD_Send_Char(VFD_Buffer[i]);
  for (byte i=Len; i < 20; i++) VFD_Send_Char(' ');

  // And on line 2 - This will be erased if the character actually displays anything.
  VFD_Send_Cmd(20);       // Force it to position 20
  for (byte i = 0; i < 20;  i++) VFD_Send_Char('A' + i);

  // Now write a line of the Ascii character to the second line
  // - Non-displaying characters do not display anything and do not move the cursor.
  VFD_Send_Cmd(20);       // Force it to position 20
  for (byte i=0; i < 20; i++) {
```

```
    VFD_Send_Char(Ascii);      // Send some good stuff to the VFD!
  };
  delay(250);
};

//CPAGE-------------------------------------------------------------------------------------
//   Display various bitmaps in a custom character.
//-------------------------------------------------------------------------------------

// Try out some custom characters. We can only have two custom bitmaps at a time.
// - We will change the bit map for 'A'
// There is a bug here. The VFD uses two buffers for character bitmaps.
// A write will always use the next buffer, even if it is re-defining a character that already
// has a bit map assigned. The implications are:
// - Redefining a single character will erase the definition for another character.
// - Here's the bug: If you redefine an overridden character, it does not update the
// - Character to use the new buffer so you don't see the update. If you define two characters
//   or re-define the same character twice it works ok.
// Bottom line is this does not work well for dynamically changing characters. If you load two
// custom character bitmaps during initialization and then use them, it will be fine.
byte Data[5];

memset(VFD_Buffer, ' ', sizeof(VFD_Buffer));
VFD_Str_To_Buffer(VFD_Buffer, "Custom bitmaps.");
VFD_Send_Full_Buffer(VFD_Buffer, 0);
VFD_Send_Char(VFD_Mode_Cursor_Off);            // No cursor!

// Create custom bitmaps with one additional bit turned on each time.
// - From this display, you can see how the bits are translated to pixels.
memset(Data,0x00,sizeof(Data));                // Clear all of the data
for (byte row = 0; row < 5; row++) {
  for (byte bt=0; bt < 8; bt++) {
    bitSet(Data[row],bt);                      // Set another bit on
    VFD_Send_Char(VFD_Set_Char_Bitmap);        // Send ESC
    VFD_Send_Char('A');                        // Send character to modify
    VFD_Send_Char(Data[0]);
    VFD_Send_Char(Data[1]);
    VFD_Send_Char(Data[2]);
    VFD_Send_Char(Data[3]);
    VFD_Send_Char(Data[4]);

    // twice because of double-buffer bug. The buffers will now both be the same.
    VFD_Send_Char(VFD_Set_Char_Bitmap);        // Send ESC
    VFD_Send_Char('A');                        // Send character to modify
    VFD_Send_Char(Data[0]);
    VFD_Send_Char(Data[1]);
    VFD_Send_Char(Data[2]);
    VFD_Send_Char(Data[3]);
    VFD_Send_Char(Data[4]);

    VFD_Send_Cmd(20);                          // Move to posn 20
    VFD_Send_Char('B');
    VFD_Send_Char('0'+row);
    VFD_Send_Char('b');
    VFD_Send_Char('0'+bt);

    // Display a few of the new characters.
    for (byte i=0; i < 10; i++) {
      VFD_Send_Char('A');
    };
    delay(250);                                // Slow it all down so we can see it.
  };
};

delay(1000);                                   // Gaze in wonder again!

//CPAGE-------------------------------------------------------------------------------------
//  Since we've got a custom bitmap will all bits on, lets light them all up at once!
//-------------------------------------------------------------------------------------
memset(VFD_Buffer,'A',40);
VFD_Send_Full_Buffer(VFD_Buffer,0);
```

```
  delay(2000);

  // Another SW reset test. All the character should be normal again after this.
  VFD_Send_Cmd(VFD_Cmd_SWReset);                // Software reset
  VFD_Send_Char(VFD_Mode_Cursor_Off);           // No cursor!

  //CPAGE---------------------------------------------------------------------------------------
  //   Change the display brightness.
  //---------------------------------------------------------------------------------------------
  memset(VFD_Buffer, ' ', sizeof(VFD_Buffer));
  VFD_Str_To_Buffer(VFD_Buffer, "Brightness: X");
  VFD_Str_To_Buffer(VFD_Buffer + 20, "ABCDEFGHIJKLMNOPQRST");
  VFD_Send_Full_Buffer(VFD_Buffer, 0);

  // Change the intensity to teh four supported value.
  // -  0, 16, 32, 48.
  // -  0 is brightest.
  // - 48 is dimmest.
  for (byte num=0; num < 4; num++) {  // Run through it a few times!
    for (byte i=0; i <= 3; i++) {
      VFD_Send_Char(VFD_Set_Brightness);        // intensity
      VFD_Send_Char(i<<4);                      // Send code (mult by 16)
      VFD_Send_Cmd(12);                         // Put 0-3 in the header
      VFD_Send_Char('0' + i);
      delay(1000);
    };
  };
  VFD_Send_Char(VFD_Set_Brightness);            // Back to full intensity.
  VFD_Send_Char(0);

}; // Loop

//CPAGE=============================================================================================
// VFD_Str_To_Buffer
//
// - Copy the string in the second parameter into Buff.
// - Note that this is not a string copy in that the termnating NULL is NOT copied, but is
//   required to be in the string.
//
//==================================================================================================
void VFD_Str_To_Buffer(char * Buff, char * Str) {

while (*Str != 0) {
  *Buff=*Str;
  *Buff++;
  *Str++;
};

};  //  VFD_Str_To_Buffer

//CPAGE=============================================================================================
// VFD_Send_Full_Buffer(char * Buff, byte CurPosn);
//
// - Here is the function to use for Displays-for-Dummies.
// - Format all the characters you want to see in a 40-byte buffer.
//   - The buffer should only contain 'printable' characters for the disaplay.
//   - No control characters, or other characters that do not display.
// - Hand it over to this function and the display will be updated appropriately.
// - The cursor mode is not changed. That is, if it was displayed, it will remain displayed.
// - The cursor is left in the position indicated by the second parameter.
// - If blinking characters are set, the blinking is not changed.
// - You should not call this function if vertical or horizontal scrolling mode is turned on:
//   - It is not possible to write a character in the last column without forcing a
//     horizontal scroll.
//   - A vertical scroll will happen if the last column on line two is written to.
// - If you need scrolling on, this routine is not for you!
// - This might seem like over kill if you are only changing a couple of characters on
//   the display, but the full refresh takes less than 5 milliseconds and makes the
//   display management easy.
//
//==================================================================================================
```

```
void VFD_Send_Full_Buffer(char Buff[], byte CurPosn) {

  VFD_Send_Cmd(0);                                // Move cursor to beginning
  for (byte i = 0; i < 40; i++) {
    VFD_Send_Char(Buff[i]);                       // Write out the full buffer
  };
  if (CurPosn > 39) {
    CurPosn = 0;                                  // Fix possible bad value.
  };
  VFD_Send_Cmd(CurPosn);                          // Put the cursor in position.

};  //  VFD_Send_Full_Buffer

//CPAGE=============================================================================================
// VFD_Send_Str
//
// - Send all of the character in a string to the VFD.
// - They are written starting at the current cursor position.
//
//==================================================================================================
void VFD_Send_Str(char * Str) {

while (*Str != 0) {
  VFD_Send_Char(*Str);
  *Str++;
};

};  //  VFD_Send_Str

//CPAGE=============================================================================================
// VFD_Send_Cmd and VFD_Send_Char
//
// - Commands are sent with AD high. Data are sent with AD low.
// - Send a command or data character to the display.
// - Direct port manipulation would be a good idea here if speed matters at all. It does not
//   for this sample application.
// - This code supports any line to be on just about any digital capable pin of the Arduino.
// - The digitalWrite's and pinMode's below each take about 4us to execute for a total of
//   about 45-50us total to run this function.
// - It could probably be done in 1-2us if D0-D7 were on a single port,and all control lines
//   on anotehr port.
// - The display is happy with sub-microsecond switching times on its pins.
//
// Specific to VFD_Send_Cmd:
//
// - The AD line must be implemented for Send_Cmd to work.
// - A parameters of 0-39 will execute a move-cursor command to place the cursor on the given
//   positon of the display. Over 39 will go to position 0.
// - A parameter of 0x40 (64) will perform a software initiated reset of the display.
//
//==================================================================================================
void VFD_Send_Cmd(byte Ch) {

  VFD_Check_Cmd_Timing();                    // Can we issue the next command yet?

  // Issue command to VFD
  digitalWrite(VFD__WR, LOW);                   //Prepare to write
  digitalWrite(VFD_AD, HIGH);                   // Set command mode
  digitalWrite(VFD_D0, bitRead(Ch,0));
  digitalWrite(VFD_D1, bitRead(Ch,1));
  digitalWrite(VFD_D2, bitRead(Ch,2));
  digitalWrite(VFD_D3, bitRead(Ch,3));
  digitalWrite(VFD_D4, bitRead(Ch,4));
  digitalWrite(VFD_D5, bitRead(Ch,5));
  digitalWrite(VFD_D6, bitRead(Ch,6));
  digitalWrite(VFD_D7, bitRead(Ch,7));
  digitalWrite(VFD__WR, HIGH); // Write Complete

  LastCmdTime=micros();                          // Rememeber when the last command was issued.

  // Set the delay until the next command can be issued.
```

```
  if (Ch == VFD_Cmd_SWReset) {
    VFDCmdWait = VFD_Delay_SWReset;
  }  else {                                   // Assume a cursor move.
    VFDCmdWait = VFD_Delay_Cursor_Posn;
  };

};  //  VFD_Send_Cmd

//CPAGE=============================================================================================
// VFD_Send_Char
//
// - See VFD_Send_Cmd for more inoformation.
//
// Specific to VFD_Send_Char:
//
// - The follwoing parameter values are supported. See the device documentation for details.
//   - 0x00 - 0x1F - Various control commands that set operation mode and can define characters.
//                 - Some commands are multi-character so they will continue to process the
//                   data that follows as parameters to the command.
//   - 0x20 - 0xff - Displays the character from the character generation rom at the current
//                   cursor position. Not all character positions are defined in the ROM.
//==================================================================================================
void VFD_Send_Char(byte Ch) {

  VFD_Check_Cmd_Timing();                     // Can we issue the next command yet?

  // Issue command to VFD
  digitalWrite(VFD__WR, LOW);                 //Prepare to write
  digitalWrite(VFD_AD, LOW);                  // Set data mode
  digitalWrite(VFD_D0, bitRead(Ch,0));        // Put the bits on the data lines
  digitalWrite(VFD_D1, bitRead(Ch,1));
  digitalWrite(VFD_D2, bitRead(Ch,2));
  digitalWrite(VFD_D3, bitRead(Ch,3));
  digitalWrite(VFD_D4, bitRead(Ch,4));
  digitalWrite(VFD_D5, bitRead(Ch,5));
  digitalWrite(VFD_D6, bitRead(Ch,6));
  digitalWrite(VFD_D7, bitRead(Ch,7));
  digitalWrite(VFD__WR, HIGH);                // Write Complete

  LastCmdTime=micros();                       // Remember when the last command was issued.

  // Set the delay until the next command can be issued.
  if (Ch == VFD_Ctl_FF) {
    VFDCmdWait = VFD_Delay_Clear;             // Clears take a lot longer!
  } else {
    VFDCmdWait = VFD_Delay_Mode_Normal;       // Assume a worst case command
  };

};  //  VFD_Send_Char

//CPAGE=============================================================================================
// VFD_HW_Reset
//
// - Perform a hardware reset command on the VFD.
// - Just pulse the reset line high for a few us.
//   - About 4.0us was enough (that's about as fast as the Arduino can toggle the line using
//     digitalWrite). But bringing it up to a total of about 10us will make sure the pulse
//     did not get sucked up by some capacatence or inductance somewhere.
// After reset:
// - The display is cleared
// - The cursor position is set to 0 (top left
// - Cursor mode DC1 is set (visible, non-flashing cursor)
// - Scroll modes are off.
// TODO:
// - How long does a reset really take? Need to monitor the Status line.
//==================================================================================================
void VFD_HW_Reset() {

  VFD_Check_Cmd_Timing();                     // Can we issue the next command yet?

  digitalWrite(VFD_RS, HIGH);                 // Pulse the reset line
```

```
  delayMicroseconds(VFD_Delay_Reset_Pulse);
  digitalWrite(VFD_RS, LOW);                    // Done with the pulse

  LastCmdTime = micros();                       // Remember when the last command was issued.
  VFDCmdWait = VFD_Delay_HWReset;               // Don't issue the next command for this long.

};  //  VFD_HW_Reset

//CPAGE=========================================================================================
// VFD_Test_Mode
//
// - Put the VFD in test mode.
//   - Steps the display through all of the characters it supports.
//   - It will stay in this mode until it is powered off or HW reset.
//   - Control sequence:
//     - _TS=0
//     - Short positive reset pulse. (4.0us was OK)
//     - Hold _TS=0 for at least 130ms. 129ms did not work. Use 150ms to be sure.
//     - _TS=1
// - The display will stay in test mode until the next hardware reset or power cycle.
//   - It displays its internal character set from 0x20-0xFF on the display at a rate of about
//     150ms/char. When its done, it just does it again.
//   - One run through the sequence takes about 28 seconds.
//===============================================================================================
void VFD_Test_Mode() {

  VFD_Check_Cmd_Timing();                       // Can we issue the next command yet?

  digitalWrite(VFD__TS,LOW);                    // Pull down the test line
  delayMicroseconds(VFD_Delay_Reset_Pulse);
  digitalWrite(VFD_RS,HIGH);                    // Call for a reset
  delayMicroseconds(VFD_Delay_Reset_Pulse);
  digitalWrite(VFD_RS,LOW);                     // Reset pulse is done
  delay(VFD_Hold_Test/1000);                    // Hold down _TS until the test starts
  digitalWrite(VFD__TS, HIGH);                  // Can release test now

  LastCmdTime = micros();                       // Rememeber when the last command was issued.
  VFDCmdWait = VFD_Delay_Test;                  // Don't issue the next command for this long.

};  //  VFD_Test_Mode

//CPAGE=========================================================================================
// VFD_Check_Cmd_Timing
//
// - Since the VFD does not return any sort of "busy" signal, we must wait a certain amount
//   of time before issuing another command to it.
// - If not enough time has passed, we will delay for the remainder of the command-busy time.
// - The time we wait is dependent on the actual command that was issued. That time is
//   set in a global whenever a command is issued.
// - The micros() function rolls over about every 70 minutes (16Mhz Arduino):
//   - It is possible that over 70 minutes has passed since a command was last issued.
//   - In a tiny number of cases we will end up inserting a few more microseconds of wait time,
//     but it won't matter as we've already been waiting a very long time.
//===============================================================================================
void VFD_Check_Cmd_Timing() {

  unsigned long     DeltCmdTime = 0;            // us since the last commnd was executed.

  DeltCmdTime = micros() - LastCmdTime;         // How long since we issued the last command?
  if (DeltCmdTime < VFDCmdWait) {               // Do we still need to wait for a while?
    DeltCmdTime=VFDCmdWait - DeltCmdTime;       // How long to wait
    if (DeltCmdTime > 15000) {                  // delaymicroseconds does not work above 16000!
      delay(DeltCmdTime/1000);                  // wait in milliseonds
    } else {
      delayMicroseconds(DeltCmdTime); // Wait the required microseconds
    };
   };

};  // VFD_Check_Cmd_Timing
```

Appendix-2 Noritake CU40026SCPB-S20A Datasheet

The following data sheet describes a very similar 80 x 2 character VFD display from Noritake. It is twice the size, so you might expect the current draw to be a lot more than the T23A and obviously the size is bigger.

There are significant differences in the command structure and capabilities that are probably attributable to the T23A device being a custom module that only includes the functions that were required for the application. The S20A might be a custom display as well.

Nevertheless a lot of the information is relevant, especially probably the electrical interface specifications.

I'm sure all of the datasheet is copyrighted by Noritake, and although it is included here, there is no attempt here to put this portion of the document in the public domain.

VACUUM FLUORESCENT DISPLAY MODULE

SPECIFICATION

MODEL :CU40026SCPB-S20A CUSTOMER

SPECIFICATION NO. :DS-31-0000-01
ISSUE DATE Sep . 21,1987

No: OF PRESENTATION

REVISION :Jan. 14.1988
:May 23,1988

PUBLISHED BY: ISE ELECTRONICS CORPORATION JAPAN

1.0 General Description

1.1 Application: Readout of computer, microcomputer communication terminal and automatic instruments.

1.2 Construction: Single board display module consists of 80 character VFD, refresh memory, character generator, control circuit and DC/DC converter.

1.3 Display color: Blue-green.

1.4 Outline dimension : See attached drawings.

2.0 Absolute Maximum Ratings

Power Supply Voltage ----- V_{CC} : +7.0 Max, V_{DC}
Logic Input Voltage ------ V_{IN} : +7.0 Max. V_{DC}

3.0 Electrical Ratings

Parameter	Symbol	Min.	Typ.	Max	Unit
Power supply voltage	V_{CC}	4 75	5.00	5.25	V_{DC}

4 0 Electrical Characteristics

PARAMETER		SYMBOL	MIN	TYP	MAX	UNIT	CONDITION
INPUT VOLTAGE	H	V_{IH}	2.4	-	V_{CC}	Vdc	V_{CC} =5.0V
	L	V_{IL}	-	-	0.8	Vdc	V_{CC} =5.0V
OUTPUT VOLTAGE	H	V_{OH}	2.4	-	-	Vdc	I_{OH} = 400uA
		V_{OL}	-	-	0.4	Vdc	I_{OL} = 1.6 mA
Supply current		I_{cc}	-	0.75	0.85	A	VCC=5.0V Operate all dots in all chr positions

Note:
Power-on delay of V_{CC} shall be with 30 ms.
I_{CC} might be anticipated more than 2 times figure of above table at power on rush.

5.0 Optical Specifications

Number of characters : 80(2 line x 40 chrs)
Matrix format : 5 x 7 dot character with
cursor Display area : 188.55 mm x 16 mm(X x Y)
Character size : 3.3 mm x 5.05 mm(X x Y)
Character pich : 4.75 mm(center-to-center)
Dot size : 0.5 mm x 0.55 mm(X x
Y) Dot pitch : 0.7 mm x 0.7 mm(X x
Y) Luminance : 700 Cd/m^2(Typ.) or
2OOfL Color of illumination : Blue-green

6.0 Environmental Specifications

Operating temperature : -20 to +60°C Storage temperature : -40 to t70°C Operating humidity : 20 to 80% R.H

7.0 Mechanical Strength

Vibration Test	:	Frequency	:10-55-10 HZ
		Sweep time	:1 minute
		Amplitude	:2 mm (Fixed 10G)
		Direction	:X,Y & 2 (3 directions)
		Times	:30 Min, for each direction
Shock Test	:	Acceleration	:100G
		Duration	:9.0 sec
		Direction	:X,Y & 2 (3 directions)
		Times	:Three (3) times for each direction

The test shall be done at no operating and no any mechanical and electrical failures should be found after the tests.

8.0 Functional Descriptions

The CU40026SCPB-S20A VFD Module will provide the functions of DATA WRITE, DATA READ, COMMAND WRITE, STATUS READ and DISPLAY RESET.

\WR	\RD	AD	\CS	FUNCTION	DIRECTION OF DATA BUS
0->1	1	0	0	DATA WRITE	HOST TO MODULE
0->1	1	1	0	COMMAND WRITE	HOST TO MODULE
1	0	0	0	DATA READ	MODULE TO HOST
1	0	1	0	STATUS READ	MODULE TO IOST

8.1 Data Write

8.1.1 Data write

This is executed at rising edge of \WR Pulse while \CS=AO=”0” and \RD=”1”. This module accepts 158 characters and 16 control codes listed in
Table 1.
Two desired fonts may be alternated into character code of 00H to FFH in Table I with ESC(18H) code. See (16) ESC.
Generally, the cursor automatically moves to right by one character position after execution of data write.
Control code are defined as follows:
The term of "CURSOR" means the writing position.

1) BS: Back Space

DC1 Mode: The cursor position is shifted to left by one character position. When the cursor is located at the left end of the bottom line, the cursor is shifted to the right most position of the top-line after execution. When the cursor is on the left most position of the top line, the cursor is shifted to the right most Position of the bottom line.

DC2 Mode: Same as DCl Mode.

2) HT: Horizontal Tab

DC1 Mode:The cursor Position is shifted to right by one character position. When the cursor is located at the right end of the top line, the cursor is shifted to the left most position of the bottom line. When the cursor is on the right most position of bottom line, the cursor is shifted to the left most position of the top line.

DC2 Mode: When the cursor is on the right most position of the bottom line, all characters on the bottom line are shifted to one line up, and the cursor is positioned to the left most position of the bottom line. At this time, all positions of the bottom line are cleared for a new line.

3) LF: Line Feed

DC1 Mode: When the cursor is shifted to the same column position of next line. When the cursor is on the bottom line, the cursor is shifted to
the same column position of the top line.

DC2 Mode: When the cursor is on the bottom line, all characters on the bottom line are shifted to the upper line, and the cursor maintains the same position of the bottom line. At this time, all-positions of the bottom line are cleared for a new line. When the cursor is on the top line, same as DC1 Mode execution will he made.

4) CR: Carriage Return

DC1 Mode:The cursor is Positioned on the left most position of the same line.
DC2 Mode: Same as DC1 Mode.

5) DC1: Normal Mode (Default Mode)

After a character is written, the position of the cursor is automatically shifted to the right by one character position. When the cursor is on the right most position of the top line, the cursor is shifted to the left most position of the bottom line. When the cursor is on the right most position of the bottom line, the cursor is shifted to the left most position of the top line.

6) DC2: Scroll Mode

After all positions of the bottom line are written, the characters written on the bottom line are scrolled up to the top line, and the cursor is positioned at the left most position of the bottom line. At this time, all characters on the bottom line are cleared for a new line. The display module automatically selects the DC1 Mode above at initial power-on time. This selection will be maintained until another mode is selected.

7) DC3: Cursor On Mode (Default Mode)

The Cursor position is displayed as an under-line.

8) DC4: Block Cursor Mode

The character on cursor Position is alternatively flickering with full dots.

9) DC5: Cursor Off mode
The under-line on cursor position is becoming invisible and DC4, DC6 modes are cancelled.

10) DC6: Cursor Blink Mode
The under-line on cursor position is flickering.

The following five control codes Select the font as follows:

11) SUB: English font (USA ASCII -7) (Default Code)

12) FS : Danish font (BCMA-7)

13) GS : General European font (ECMA-7)

14) RS : Swedish font (ECMA-7)

15) US : German font (ECMA-7)

Conversion Table from ASCII to ECMA is shown as follows:

HEX CODE	CONVERSION CODES				
	1R	1C	1D	1E	1F
23					
5B					
5C					
5D					
5E					
7B					
7C					
7D					
7E					
	RSCII	DANISH	GEN EUROPE	SWEDISH	GERMAN

SUB(1AH),English font, is automatically selected at the power-on reset. Theselected mode is maintained unless other mode is selected.

16) ESC:

The following ESC code assigns two iser desired fonts (UDF) into any character positions from 00H to FFH of table 1. RAM of the module reserves two-Character-size of memory for these new characters. Six-byte data succeeding this ESC code alternates present character font to new font desired.

1st byte:
1BHEX
2nd byte:Definition character code
Definable character codes are available from 00H to FFH of table 1. If the character code of control characters as
BS,HT, CR, etc, is selected for a new character, the module displays new character instead of control action. Caution that definition of 1BH(ESC) character code kills ESC function thereafter.
3rd--7th byte : Formation of character font
Each dot data of 5 x 7 is defined with following Table. Figures In the Table are corresponded to each dot position of
5 x 7. The dots to be lighted shall be specified as "1" (active high).

BYTE	D7	D6	D5	D4	D3	D2	D1	D0
3rd	23	15	22	16	21	17	20	18
4th	27	11	25	12	25	13	24	14
5th	31	9	30	*	29	*	28	10
6th	35	5	34	6	33	7	32	8
7th	*	8	*	2	19	1	UL	4

***="O" (low) UL:Under line**

After execution of above sequence, the new character defined will be displayed by defined character code.

DISPLAY DOT

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30
31	32	33	34	35

Definition of new character "!" to Character code AO Hex:

5 X 7
DOT Dot
pattern

		0		
		0		
		0		
		0		
		0		

5 X 7 Dot

Specify each dot

Byte/Bit	7	6	5	4	3	2	1	0	HEX
3rd Byte	0	0	0	0	0	0	0	1	01
4th Byte	0	0	0	0	0	1	0	0	04
5th Byte	0	0	0	0	0	0	0	0	00
6th Byte	0	0	0	0	1	0	0	1	09
7th Byte	0	1	0	0	0	0	0	0	40

Then Syntax should be written: 1B + A0 + C1 + 04 + 00 + 09 + 40 (Hex)

8.2 Command Write

This is executed at rising edge of \WR pulse while \CS = "O" and A0=\RD=”1”. This module provides following commands:

00XX XXXX: Set the cursor on 00XX XXYM(HEX) Position.
0000 0000 (00 Hex) : The left most of the top line.
0000 0001 (01 Hex) : The 2nd column of the top line.
0010 0111 (27 Hex) : The right most of the top line.
0010 1000 (28 Hex) : The left most of the bottom line.
0100 1111 (4F Hex) : The right most of the bottom line.
When more than the number of characters (80) is specified, the cursor will not move.

0101 0000: (50H) Software reset
The same execution as hardware reset of Part 8.4

0101 0001: (51H) Read the cursor position.
Read data 00H means the left most cursor position.

8.3 Data Read

After the data read command (51H) is written, data read is executed when \CS=A0=\RD=”0” and \WR=”1”. After confirming that bit 0 of status data is
“1”
No confirming of status data, however, is needed, only when the data read is executed After 1.0 ms or more of command write.

8.4 Status Read

The module outputs the status on bit 1 of data bus, when \CS=\RD=”0” and A)=\WR=”1”
BIT 0: Status of data read: data read is valid when BIT 0=”1”
BIT 1: Status of data write: data write and command write are valid only when BIT 1="O".
BIT 2 through 7: Do not care.
No confirming of status bits, however, is needed, only when the period of write cycle is longer than 1.0 ms.

8.5 Hardware Reset

RESET"1" makes the module initialized as follows:
1. All character Positions are filled with SP(20H) characters.
2. The cursor position is set on the left most position of the top line.
3. DC1 and DC3 modes are selected.
4. Alternated characters specified by ESC code are cancelled and standard characters in character generator are selected.

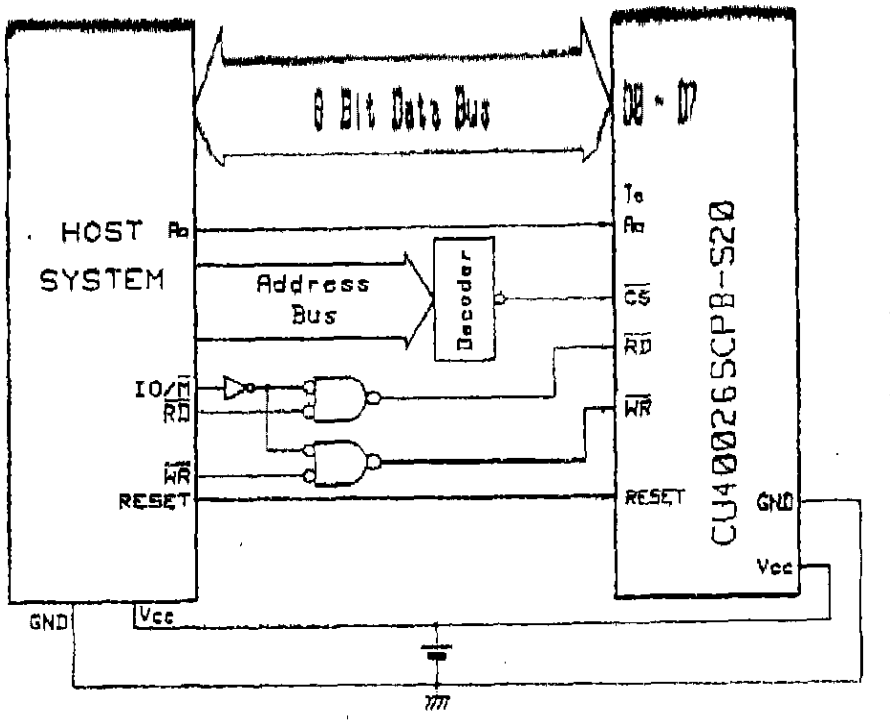
Reset signal is active high and shall be maintained 50 ms or longer. No input is executed within 100 ms after reset pulse or reset command. (See TIMING CHART)

8.6 Test Mode

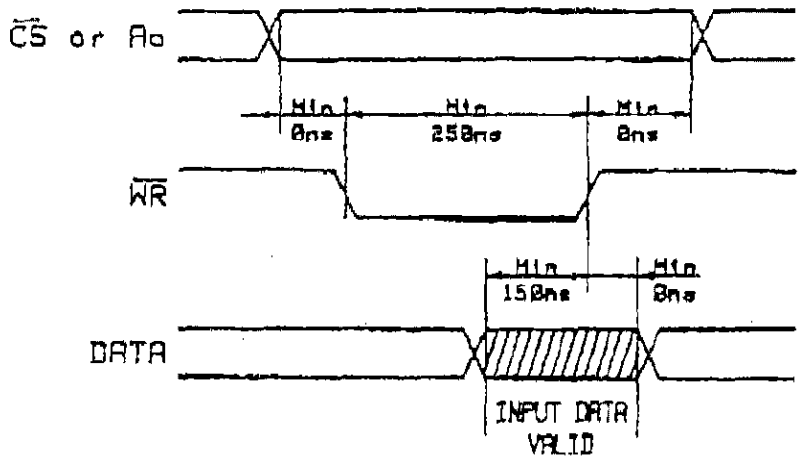
"0" more than 100msec to the T0 line at the power on or reset may intialize the test mode. During the test mode, no data/commands are acceptable. The test mode can be cancelled only power off or reset at open of T0 line.
All stored ROM character fonts are displayed automatically at this mode.

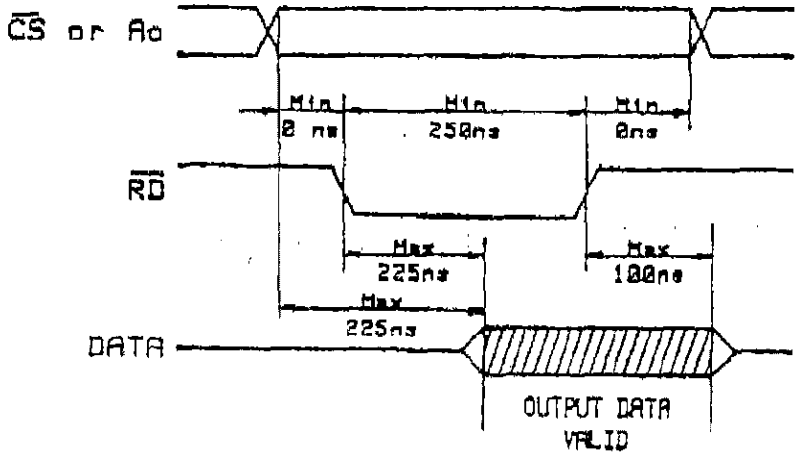
D7 D6 D5 D4		0 0 0 0	0 0 0 1	0 0 1 0	0 0 1 1	0 1 0 0	0 1 0 1	0 1 1 0	0 1 1 1	1 0 0 0	1 0 0 1	1 0 1 0	1 0 1 1	1 1 0 0	1 1 0 1	1 1 1 0	1 1 1 1
D3 D2 D1 D0		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0 0 0 0	0			SP	0	@	P	`	p				—	代	田		
0 0 0 1	1		DC1	!	1	A	Q	a	q			。	ア	チ	由		
0 0 1 0	2		DC2	"	2	B	R	b	r			「	イ	ツ	絵		
0 0 1 1	3		DC3	#	3	C	S	c	s			」	ウ	テ	下		
0 1 0 0	4		DC4	$	4	D	T	d	t			、	エ	ト	中		
0 1 0 1	5		DC5	%	5	E	U	e	u			・	オ	ナ	口		
0 1 1 0	6		DC6	&	6	F	V	f	v			ヲ	カ	ニ	目		
0 1 1 1	7			'	7	G	W	g	w			ァ	キ	ヌ	力		
1 0 0 0	8	BS		(	8	H	X	h	x			ィ	ク	ネ	久		
1 0 0 1	9	HT		)	9	I	Y	i	y			ゥ	ケ	ノ	女		
1 0 1 0	A	LF	SUB	*	:	J	Z	j	z			ェ	コ	ハ	子		
1 0 1 1	B		ESC	+	;	K	[	k	{			ォ	サ	ヒ	口		
1 1 0 0	C		FS	,	<	L	¥	l	|			ャ	シ	フ	力		
1 1 0 1	D	CR	GS	-	=	M	]	m	}			ュ	ス	ヘ	大		
1 1 1 0	E		RS	.	>	N	^	n	~			ョ	セ	ホ	中		
1 1 1 1	F		US	/	?	O	_	o				ッ	ソ	マ	日		

9.0 Interface Example

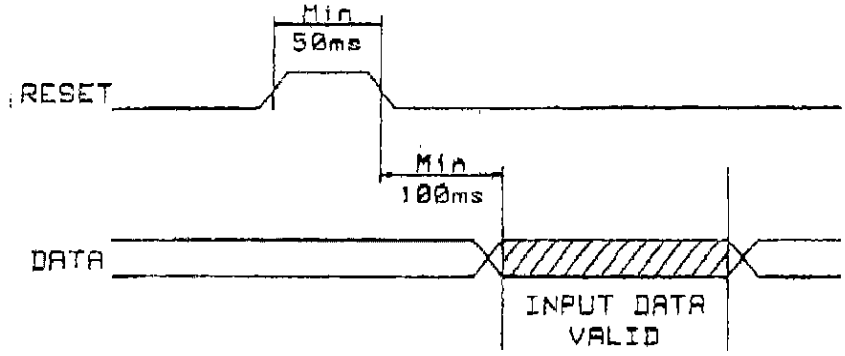


10.0 Data Write/Read Timing

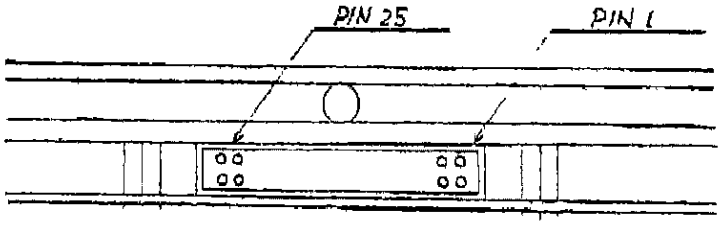




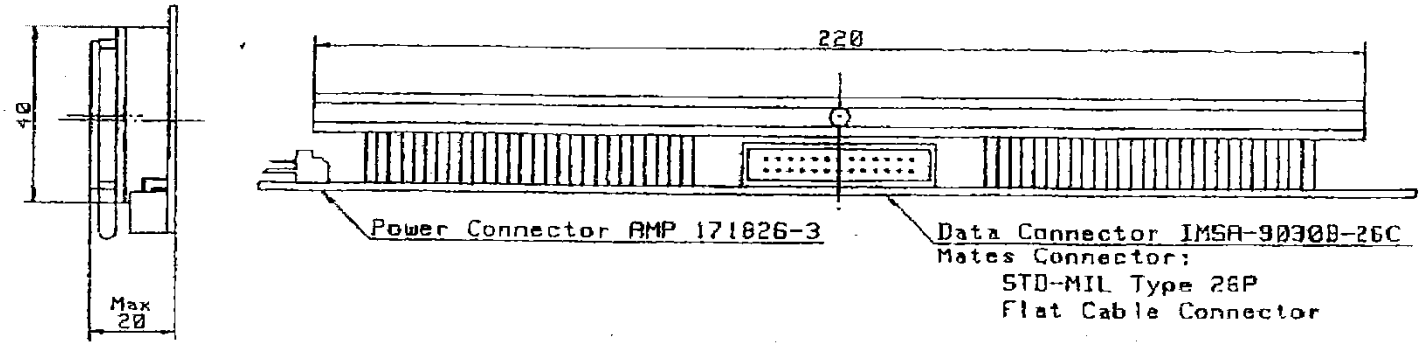
11.0 Reset Timing



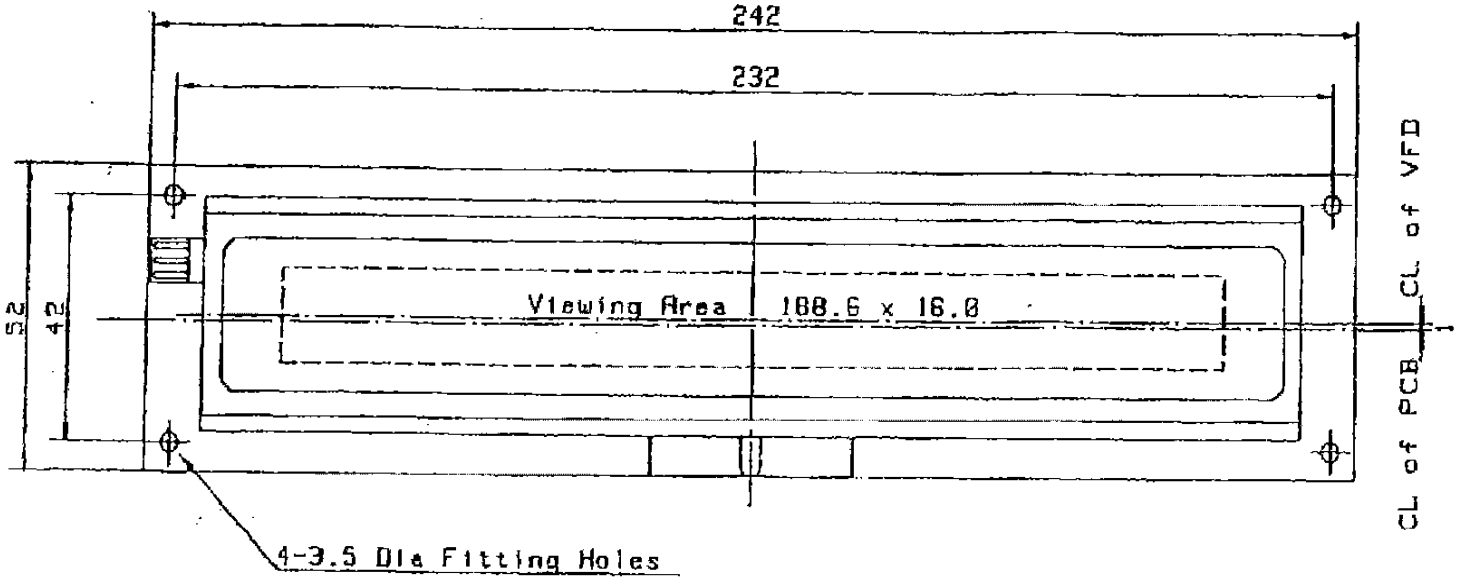
12.0 Pin Connection



PIN NO	SIGNAL	PIN NO	SIGNAL
1	D2	2	GND
3	D6	4	GND
5	05	6	GND
7	D4	8	GND
9	D3	10	GND
11	D2	12	GND
13	D1	14	GND
15	D0	16	GND
17	\WR	18	GND
19	A0	20	RESET
21	\RD	22	GND
23	\CS	24	GND
25	T0	26	GND



00265CPB-S20 OUTLINE DIMENSION



IMPORTANT PRECAUTIONS

*** All VFD Modules contain MOS LSI's or IC's. Anti-Static handling procedures are always required.**

*** VF Display consists of Soda-lime glass. Heavy shock more than 100 G, thermal shock greater than 10°C/minute. Direct hit with hard material to the glass surface especially to the EXHAUST PIPE -- May CRACK the glass.**

*** Do not PUSH the display strongly. At mounting to the system frame, slight gap between display glass face and front panel is necessary to avoid a contact failure of lead pins of display. Twist or warp mounting will make the glass CRACK around the lead Pin of display.**

*** Neither DATA CONNECTOR or POWER CONNECTOR should be connect or disconnected while power is applied.**
As is often the case with most subsystems, caution should be exercised in selectively disconnecting power within a computer based system. The modules receive high logic on strobe lines as random signals on all data Ports. Removal Of Primary Power with logic signals applied may damage input circuitry.

*** Stress more than specification listed under the Absolute Maximum Ratings may cause PARMANENT DAMAGE of the modules.**

*** +5 volts Power line must be regulated completely since all control logics are depended on this line.**
DO not apply slow-start power. Provide sufficient output current power source to avoid trouble of RUSH CURRENT at Power on. (At least output current of double figure of Icc. listed on the specification of each module, is required.)

*** Data cable length between module and host system is recommended within 500 mm to free from a mis-operation caused by noise.**

*** Do not place the module on the conductive plate just after the power off. Due to big capacitors on the module, more than 1 min. of discharging time is required to avoid the failure caused by shorting of power line.**