NOKIA Graphics LCD Board

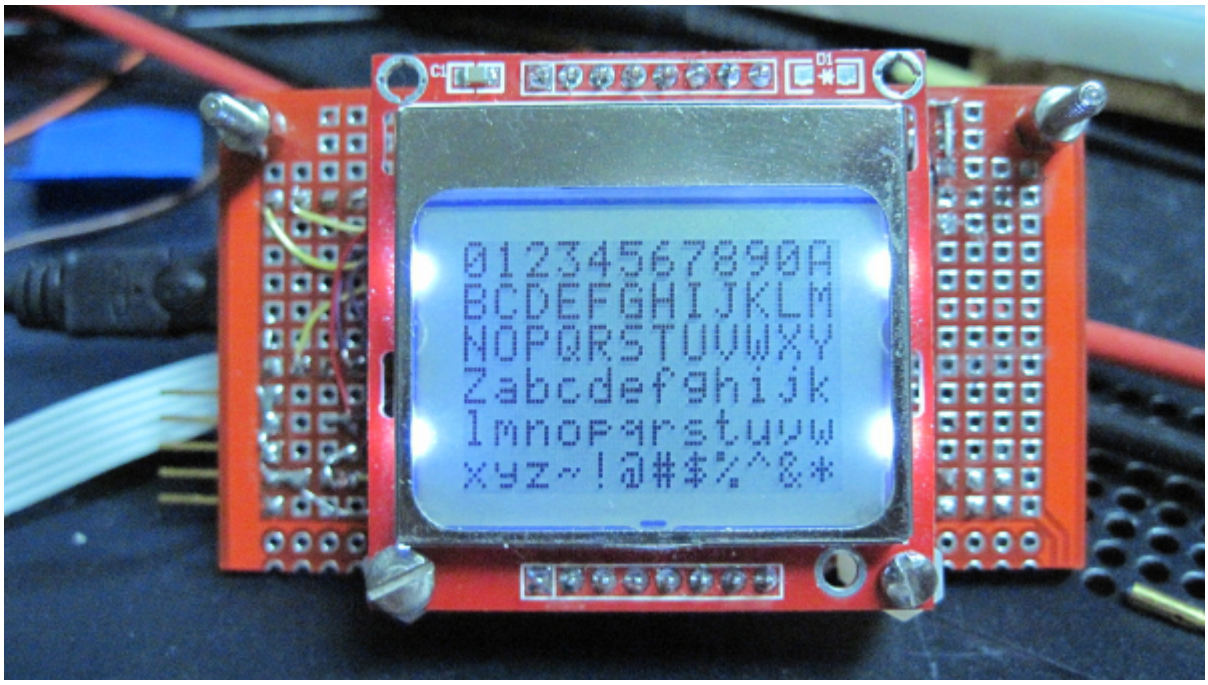

Photo 1. Nokia Development Assembly

This manual describes a demo application for the Nokia 5110 graphics LCD display module (Nokia module). The Nokia's monochrome graphics display area is 48 by 84 pixels. The demo application shows how to implement two character sets, "Normal" and "Big." The demo also shows how the Normal and Big characters can be placed at specified locations on the display.

In the following, the demo is called the "NOKIA" application. Photo 1, above, shows the NOKIA project development assembly built on a 3 by 1 3/4 inch piece of perforated breadboard and wired point to point with wire-wrap wire. The display shows Normal size characters.

Depending on a jumper setting, the NOKIA project board starts up in either a serial command mode or a demo display mode. The serial command mode allows the display to be exercised using commands sent from a terminal program or a control microprocessor.

In the demo display mode, the display shows the the current version number, display settings, two character sets and some special effects.

The NOKIA project was originally designed as a command-driven peripheral to offload display processing from another processor. This uses only the two asynchronous serial lines to provide a display instead of the multiple I/O lines that are typically needed to directly drive a display.

(BLANK PAGE)

Contents

Figures

Introduction

General

The NOKIA project is based on an Arduino Nano board and a Nokia 5110 graphics LCD module. It is primarily intended to demonstrate character I/O on the display. A schematic and photo of the Nano and the Nokia display assembly provides construction information.

The NOKIA project includes an option jumper to start the board in the demo or serial command modes. In the command mode, serial commands from a terminal exercise the display features such as changing backlight intensity or displaying characters.

Because the display only needs five SPI and control lines, there are unused Nano I/O pins that can be used to expand the project into a more capable standalone application. Unused I/O is noted in Schematic 1.

Thus, the NOKIA demo can serves as a starting point for an application that requires a text/graphics display.

This manual describes commands that can be executed either from a terminal program running on a PC or directly over the serial port by a host processor (e.g., another Nano or Raspberry Pi). In the following it should be understand that commands can be entered using either of these two methods.

The manual also describes how to augment the commands and employ unused hardware for application extensions.

Unused I/O includes Port D, Pins 2 and 3 (PD2 and PD3), which are generally reserved for external interrupts 0 and 1. These pins can connect to an incremental encoder (IE), as described at the myforth-rjn archives at github.

Software

The source code for this application is at github. The source provides implementation details, command encoding and other information related to the demo. The source is provided to assist in extending the command repertoire and for use as a basic display driver.

The code for the Nano was developed with Charley Shattuck's ***myforth-arduino*** system, also at github. ***myforth-arduino*** system is primarily

intended to be used as a Forth-like macro assembler. The system provides useful features of Forth, such as looping constructs and a stack, to be used when they make sense. But, ***myforth-arduino*** is used most effectively when it is used as a macro-assembler that takes advantage of the versatile intstruction set of the Atmel 328p processor.

Introduction (Cont.)

Using ***myforth-arduino*** results in a relatively small system. The demo takes slightly more than 7K bytes of Flash memory. This includes a standalone command interpreter. The compiled image also includes a number of utilities, such as math routines, that are generally useful in standalone applications but are mostly unused in the demo.

Forth Features

Because the NOKIA demo application is intended for use with a project based on ***myforth-arduino***, many Forth-like features are used to executing commands.

The most important feature of Forth command entry is the use of the stack to pass parameters to commands requiring them.

To enter a value on the stack, first ensure the applicable number base, either **hex** or **decimal** is set. Once the numeric base is set, it remains in effect until it is changed. The stack value is entered by sending the ASCII-coded numeric value to the serial port, either by typing on a terminal or transmitting the bytes over the serial port. The stack can accept 16-bit values although most NOKIA parameters are a single byte.

For example, to put the hexadecimal byte 0xca on the stack, enter: **hex ca** .

When displaying stack values on a terminal or transmitting them back to a host processor, you can use the commands **.** (dot) and **.s** (dot "s"). The **.** command shows the item on the top of stack, in the current numeric base, removing it in the process. The **.s** command non-destructively shows all stack items. The commands **h.** and **u.** are also availble to show stack items as hexadecimal and unsigned integers, respectively.

In the following sections, it is assumed that the reader is familiar with the

above commands. Note the the "dot" command is somewhat difficult to see but should be readable in command examples that use bold characters.

Future

The author is currently working on a printed circuit board (PCB) that supports the display module, the Nano and the level converter chip. The PCB should be somewhat smaller than the prototype (e.g., a finished size of approximately 2 by 2 inches). Contact the author (see last page of this manual) if you are interested in purchasing a board or semi-kit

---

Quick Start

This section describes commands that can be executed to illustrate basic NOKIA features.

First, ensure that the option switch is in the "command interpreter" position (the "cmd mode"). Verify this by resetting the Nano processor and observing that the startup sequence ends with a blank display and not a repeating cycle of demo screens (the "demo mode"). Both the demo and command modes include a display that shows the current options. This display should show "cmd option" on the third line.

If the NOKIA starts up in the demo mode, change option jumper to the opposite position and reset the Nano.

Next, set up a serial terminal connection to the NOKIA. This requires the connection of a USB cable to the Nano and execution of a serial terminal program on the PC. Configure the terminal for a baud rate of 57K and enable the "local echo" option. To test the terminal, press the Enter key and verify that an "ok" response is visible on the terminal's display.

One easy test of the command mode is displaying a full screen of Normal size characters. To show this, enter: **nstring** . To display a full screen of Big characters (4 times Normal), enter: **bstring** . These are two of the displays shown in the demo mode.

Here are a few more commands to try:

- **clear** - Erases the Nokia display memory and positions the cursor at

"xy" position (0,0).

- **blank** - Blanks the display but does not erase display memory. To restore the display, enter the **normal** command.
- **inverse** - Displays characters in the inverse mode (dark background, light characters). To restore, use the **normal** command.
- **all-on** - Turns on all display pixels. This does not affect display memory. To restore the previous display, use the **normal** command.
- **nset** - Displays Nokia settings such as version, contrast, option and backlight intensity.
- **go** - Starts up the demo. The only way to recover from this command is to press the reset button on the Nano (or cycle power to the NOKIA board).

Other commands are somewhat more complex to set up, requiring mode settings or parameter entry. Please review the *Forth Features* subsection in the *Introduction* above for more information on parameter entry and display.

Features

Module Assembly

The assembled NOKIA project board is approximately 3 by 1 3/4 inches and easily accommodates the Nano processor, the Nokia display module, a level converter chip and miscellaneous small components.

The prototype was deliberately oversized to easily accommodate potential project components; a custom PCB board should be somewhat smaller.

Note that the Nano is mounted on the back of the board assembly so that its reset switch, USB connector and programming header are easily accessed.

Serial Commands

The project board, when brought up in the serial command mode, can control the display with simple commands. The command mode can also be used to provide information on the terminal display or transmit data back to a PC script, program or host processor. Different command forms are available to control the display and to collect status via the serial port.

Commands that only affect the display typically have an "n" prefix. For example, the **nver** command shows the current version on the display.

Commands that are intended to echo a value back to the terminal or host processor do not have an "n" prefix. For example, the **cver** command echos the current version to the serial output but does not affect the Nokia display.

As noted in the *Introduction*, the NOKIA program maintains a stack that is used for temporary parameter storage for commands or to retain a value for later retrieval or display. Parameters are entered on the stack by simply typing them on terminal or sending them to the Nano's serial port.

For example, putting the number "123" on the stack can be performed by typing "123" on the keyboard or transmitting the ASCII character sequence "123" to the serial port. ***Note that commands are "entered" when they are followed by either a space or carriage return***.

Stack items are observed or echoed back on the serial port using the command: **.s** ("dot s"). This command is "display only" and does not affect the value(s) on the stack.

Use the **.** ("dot") command to display a stack value or send it via a serial connection. To use the "dot" command, type it on the terminal or transmit a "period" character, followed by a space or carriage return, on the serial port. The "dot" command removes the value from the stack.

Features (Cont.)

Display Formatting

Commands are available for positioning the display's cursor, specifying normal (Normal) or large size (Big) characters and displaying numeric

data. Presently, there are few graphics commands because the intent of the demo is primarily to provide basic character and numeric display capabilities. A "fill" command is available, however, to display patterns.

Spare I/O

More than a full port of unused I/O is available for other user functions (e.g., for an extended application). Unused I/O includes the external interrupt pins, INT0 and INT1, that can be connected to an incremental encoder. See the IE demo driver at github.

Level Conversion

The Nokia display module requires a 3.3 Volt supply. Thus, the the 5 Volt outputs from the Nano must be translated into 3.3 Volt signals. The NOKIA design includes a level converter chip. Three of the level converters are unused and can be used for applications requiring 3.3 Volt logic. The Nokia backlight LEDs are driven directly from a Nano PWM output and do not require level conversion.

Command and Demo Modes

An option jumper on the NOKIA assembly can be set so that the processor starts up in either the serial command mode or in the "demo" mode. The demo mode is a turnkeyed application that continually runs a demo. The demo can be invoked from the command mode by entering: **go** . Once the demo starts, it can be exited only by resetting the Nano.

SPI I/O

The Nano's SPI I/O is available on the Nano's standard 2 by 6 IDC programming header. This header, when not being used for hardware programming, can be daisy-chained to other SPI devices.

The Nokia display is managed with a hardware chip select, as with the normal SPI protocol. The Nano's SPI Master's input, MISO, is not used and may be configured as a chip select input for another SPI device. Unused I/O pins can be used as device select lines for other SPI devices.

Features (Cont.)

Power

Because the Nokia display draws under one milliampere, it is powered from the 3.3 Volt output from the Nano. The Nokia's display LEDs are driven directly from a 5 Volt Nano output and consume approximately 6 milliamperes when the backlight is at maximum intensity.

Power for the NOKIA project can be furnished from the Nano's USB connection or from the 5 Volt input pins of the programming connector (i.e., pins 2 and 4). Because there is a Schottky diode between the programming power and the USB power, both can be connected at the same time without harm.

This "dual power" configuration is ideal for development because a hardware programmer can be used to program changes to the chip and a serial terminal can be attached at the same time to exercise the NOKIA with the command interpreter. This considerably expedites development with ***myforth-arduino*** by allowing connections to be exercised by simply switching windows on the PC's display.

Documentation

This manual is available in both HTML and PDF format. The HTML format has been retained in the event that the documentation is put on a web site. It also provides separate access to graphic files. These files are stored in the ***nokia_files*** directory. A separate folder under this directory, ***HR***, provides high-resolution versions of the photos used in the HTML document.

The page numbers primarily apply to the HTML document, which can be printed with the proper page breaks. To see what the printed HTML will look like, most browsers support a "print preview" option.

The PDF format is more convenient for distribution as a standalone document. It includes live links to provide some of the hypertext capabilities of the HTML document.

The PDF manual is produced by importing the the HTML document to LibreOffice Writer and exporting it as a PDF. Before exporting, some adjustments are made to improve readability, but this may not be entirely consistent (or successful). My PDF viewer shows the manual full-page with (mostly) correct page numbers.

Source code documentation provides additional information and examples for building an extended application or for expanding the demo's command set to build a general-purpose serial processor.

Contrast

Introduction

One weakness of the Nokia displays is that a single contrast setting cannot be used for all displays. This is because the contrast may vary based on how tightly the display is clamped to the carrier PCB and the temperature.

The solution to this problem is to allow the user to adjust the contrast setting without making the process too complicated. To do this, the NOKIA system allows for contrast adjustment when new software is loaded. After initial setting, the contrast can be re-adjusted with commands that invoke the startup adjustment process or that directly write a new contrast value.

Like most LCDs, the Nokia displays are temperature sensitive. The 5110 chip has registers to set temperature compensation values, but this feature is not implemented in the demo.

Initialization

Contrast adjustment is done when the NOKIA program is initially loaded into the Nano and the display starts up. This adjustment can also be initiated from a command line.

With a new software load, eeprom memory is initialized and the contrast memory is set at 0xff. The contrast adjustment process replaces this initial value with a more usable startup setting.

To set the eeprom contrast value, the display shows a sequence of values that correspond to the contrast setting being used for the current display.

As shown in Photo 2, an on-screen prompt instructs the user to save the desired setting by pressing the reset button. As shown, the two-digit hexadecimal contrast value is displayed in Big characters that decrement every few seconds while waiting for a user action.

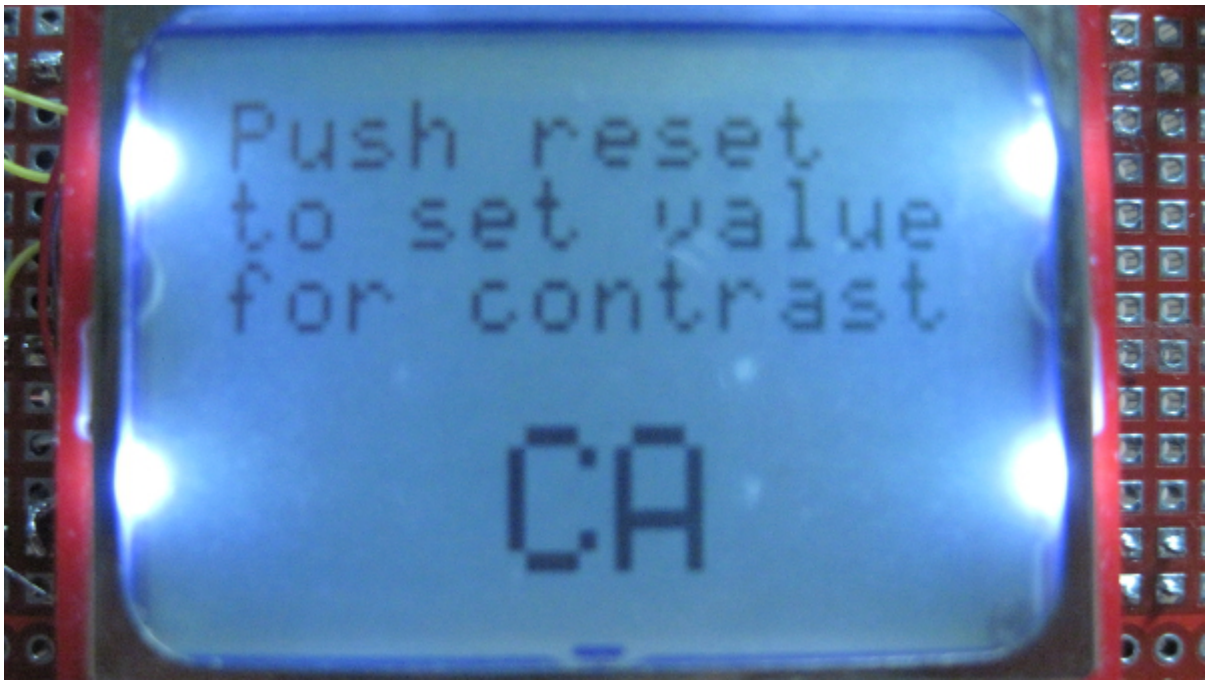

Photo 2. Contrast Setting Screen

Contrast settings range from "very dark" to "very light" with most common settings in the range of 0xd0 to 0xc0 (0xca is typical setting). If the user does not choose a startup value, the lightest contrast setting is used.

To change a contrast setting, the serial command mode must be used to issue commands to restart the initialization sequence, as described in the next section.

Commands

To restart the contrast setting sequence, enter: **new-contrast init-contrast**.

The **new-contrast** command sets the eeprom value for the contrast to 0xff so that the system starts up as it would if the eeprom were initialized.

The **init-contrast** command, when it detects an initialized contrast setting, allows the user to set the contrast interactively, as described in the Initialization section above and shown in Photo 2.

A contrast setting, such as 0xca, can be written directly to the contrast memory with the **!econ** command. For example: **hex ca !econ** sets the contrast eeprom to a value of 0xca. The **hex** command sets the serial interpreter to the hexadecimal entry mode and the **ca** entry puts the contrast parameter on the stack. The **!econ** ("store contrast") command then takes the contrast byte from the stack and writes it to the contrast memory.

Use the **!ncon** command to directly set the contrast; this does not affect the eeprom contrast setting. This command requires a parameter byte on the stack.

Current Setting

To put the current eeprom contrast setting on the stack, enter: **@econ**. To display this value on the terminal, use the **.** (dot) command. To display the contrast value without removing from the stack, use the **.s** command; this shows all of the parameters on the stack but, in this case, it non-destructively displays the contrast parameter.

To ensure that the contrast setting is displayed in hexadecimal, enter the **hex** command before displaying it with the **.** or **.s** commands. For example, enter: **hex @econ .s** . Without setting the hex entry/display mode, a value of "211" may be displayed instead of the hexadecimal "B1" value.

To set the display's contrast from eeprom, enter: **set-contrast**. This reads the contrast setting from eeprom and sends it to the Nokia. This command should be used after saving a new contrast setting to eeprom so that the display is shown with the new eeprom setting. Otherwise, the setting does not take effect until the system is reset (e.g., when the Nano's reset button is pressed).

To directly store a contrast parameter to eeprom, use the **!econ**

command. For example: **hex ca !econ** .

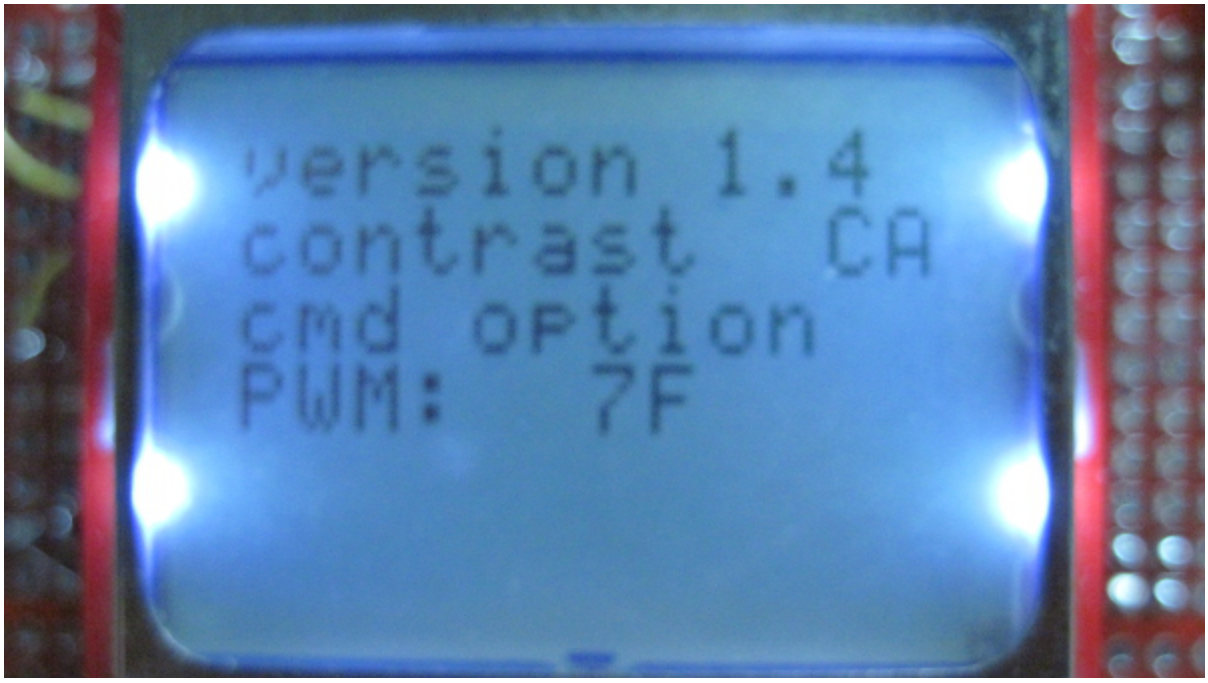

Photo 3. Settings Screen

The current eeprom contrast setting is displayed in a "settings" display at startup. Photo 3 shows the settings display. To display this screen from a command line, enter: **nset** . Commands that start with an "n" prefix typically display something on the Nokia screen instead of the terminal. To display just the contrast setting, use: **ncon** .

Backlight

Introduction

The intensity of the Nokia's backlight LEDs is directly controlled by a pulse-width modulated (PWM) output from the Nano. This allows the intensity to be continuously controlled with a byte parameter that specifies the PWM value. Because it is difficult to distinguish between levels that are close in value to each other, commands are available for common levels such as on, off, low, medium and high.

Initialization

The backlight setting is initialized from eeprom at startup. For a newly-downloaded system, the backlight will initialize to 0xff, corresponding to the **bn** (full on) setting.

The backlight level can be changed during the execution of an application with the commands given above. To save the current level to eeprom for use at startup, enter: **save-backlight**. To set the backlight from eeprom, use: **set-backlight** .

Commands

The backlight level can be set from a terminal. The most common commands are: **bf**, **bl**, **bm**, **bh** and **bn**, which correspond to backlight off, low, medium, high and on, respectively.

Set the backlight to a specific level with the **bs** (backlight set) command. For example, to set the backlight to a value between medium (0x7f) and high (0xc0), enter: **hex 9a bs**.

The above example executes the **hex** mode command before specifying a hexadecimal value (i.e., 0x9a). The hex mode remains in effect until it is changed by the **decimal** command.

Settings

Put the current eeprom setting for the backlight on the stack with: **@ebl** . This value can be displayed or sent to a host microprocessor with the "dot" command, as described above. For example, to display the current eeprom setting in hexadecimal: **hex @ebl .** .

To change the current eeprom setting for the backlight, put the desired level on the stack and execute the command: **!ebl** ("store eeprom backlight"). The eeprom backlight value is set at startup, but *the current backlight value is unaffected* when using the **!ebl** command.

After setting the backlight, the result can be verified with **@ebl**, as described above. For example, to set and verify the backlight level stored in eeprom: **hex 3f !ebl @ebl .** .

As shown in Photo 3, the current backlight setting is displayed at startup. This startup screen can be re-displayed with the command: **nset** . To display only the backlight value, use: **npwm** .

Use the **pwm** command to put the current backlight level (PWM) setting on the stack. This value can be displayed with the "dot" or **.s** commands, as described above in the Contrast section. Note that *this command gives the current backlight setting*, not the setting stored in eeprom.

Alerts

The **ba** and **bas** commands alert the user by rapidly flashing the backlight. The **ba** command is a basic standalone alert with a short duration. The demo application performs a **ba** alert just before displaying the current version.

A custom alert duration can be built using a stack parameter and the **bas** command. For example, for a longer alert than the **ba** command offers, enter: **decimal 20 bas** .

Text

Introduction

The NOKIA application provides basic character and numeric display. Characters can be displayed in two different sizes: Normal and Big. This section describes character display; a later section describes numeric display.

Text (and numeric) characters can be displayed in inverse and normal modes. Also, the screen can be blanked to hide screen data. A command to display all pixels is also available.

Characters can be positioned at specified "xy" positions on the screen. Characters output in excess of line or screen limits will wrap to the next line or to the top of the display.

Display Modes

Display modes applying to an entire screen are set with the following commands: **normal**, **inverse**, **all-on** and **blank**.

The **normal** command is the most often used and many character display words automatically execute it to ensure characters are displayed with dark pixels on a light background.

The **inverse** command reverses the screen display to show characters on a dark background with light pixels. The demo application shows a full screen of both Normal and Big characters in the **inverse** mode just after displaying the screens in normal mode.

The **all-on** command turns on all screen pixels for testing or special displays. The **blank** command blanks the display but does not affect the display memory; executing the **normal** command restores a blanked display.

The demo application shows the **all-on** and **blank** modes just after displaying Normal and Big characters in the **inverse** mode.

Positioning

Use the **!x**, **!y**, **!xy** and **0xy** commands to specify the "xy" position of screen data. The "!" in these commands is a symbolic shorthand for "store." The commands use a byte value or a byte pair stored on the stack.

For example, enter: **decimal 24 !x** to set the "x" position to 24 pixels. The range for "x" is the width of the display (0 to 83). Values in excess of the maximum width are wrapped to the following line or to the start of the screen. The "x" parameter can be thought of as a column designator. The "x" mnemonic was used instead of a column-based mnemonic because columns generally pertain to characters and are very different when characters can be displayed in two different sizes.

The **!y** ("store y") command sets the "y" position from a stack byte parameter. The parameter can range from 0 to 5 for Normal characters. The same range applies to Big characters but, because the character size is doubled, typical positions for Big characters are 0, 2 and 4.

To set both "x" and "y" values with a single command, put the position parameters on the stack and use the **!xy** command. For example, to position to "xy" coordinates (24,2), enter: **decimal 24 2 !xy** .

The above commands can be used within ***myforth-arduino*** definions by putting values on the stack (at run time) with the "#," word. For example: **: myposition 24 #, 2 #, !xy ;** is a word (command) definition that puts 24 and 2 on the stack at run time for use by the **!xy** word. When **myposition** is executed from a command line, the "xy" position is set to (24,2).

The **0xy** command positions to "xy" coordinates (0,0). The **clear** command clears the display memory, sets normal mode and positions to coordinates (0,0). To erase just the character memory, use the **erase** command.

Read the current screen position with the **@x** and **@y** commands. The "@" in these commands is shorthand for "fetch." These commands put the value of the specified coordinate on the stack. They can be displayed with the **.** (dot) or **.s** commands.

Normal Characters

To display a single Normal size ASCII character at the current position, use the **nchar** command. For example, to display a "K" at position (0,0), enter: **clear decimal 75 nchar** . After executing this, you can display another character such as "E" at the following character position with: **69 nchar** .

Normally a program displays character strings from within a Forth definition. This can be done with **ntype** . This word requires a string pointer to be on the stack. For example, here are definitons used to display **normal** and **inverse** mode characters on the demo screen:

```
\ --- string test (zero appears at 0xy -- wrapped)
-: (nstring)  s" &1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ" ntype
              s" abcdefghijklmnopqrstuvwxyz~!@#$%^&*0"  ntype ;
 : nstring   clear  (nstring) ;
 : ~nstring  clear inverse (nstring) ;
```

Note that there are enough characters in the two strings to cause the

display to wrap. Thus, the "&" in the first string is over-written with a "0." With a Normal character size, up to 12 characters fit on each line and up to 6 (0-5) lines fit on a screen (72 characters total).

Because string display definition uses the string-compiling word **s"**, characters are put in flash memory at compile time and thus consume flash memory in the program image.

Big Characters

Big characters are four times as large as Normal characters (twice as wide and twice as high). To output a single Big ASCII character to the display at the current position, use the **bchar** command.

For example, to display a "K" at position (0,0), enter: **clear decimal 75 bchar** . After executing this, you can display another Big character, "E", at the next character position on the display with: **69 bchar** .

A total of seven Big characters can be displayed on each display line. Note that the position for the first Big character on the first line is (0,0) but the position for the first Big character on the second line is (0,2).

For example, to display a Big "X" under characters on the first line, enter: **0 2 !xy decimal 88 bchar** . A total of 21 Big characters fit on a screen.

Normally a program displays a character string from within a Forth definition. This can be done with the **btype** word after putting a string pointer on the stack. For example, here is a demo definiton to display Big characters on the screen in **normal** and **inverse** modes:

```
  \
  \ --- Big string test
  -: (bstring)  s" *123456789ABCDEuvwxyz0" btype 0spi ; \ 0 appears at
0xy
  : bstring  clear (bstring) ;
  : ~bstring  clear inverse (bstring) ;
```

Note that there are enough characters in the Big string to cause the display to wrap. Thus, the "*" in the string is over-written with a "0" character.

Numbers

Introduction

It is often necessary to display numeric values. For example, ***myforth-arduino*** provides the **h.**, **ud.** and **u.** words to display numbers in hexadecimal, unsigned double (32-bit) and unsigned single (16-bit) formats, respectively. These words use pictured numeric output to assemble strings in RAM memory at the PAD location, as is standard with most Forth systems. These words (commands) can be used to show stack numbers on a terminal.

You can display numbers in these same formats in both the Normal and Big character formats. The words to display a number on the stack in Normal and Big formats are:

- **nh.** - Display a Normal-format number in hexadecimal format.
- **bh.** - Display a Big-format number in hexadecimal format.

- **nud.** - Display a Normal-format double number (32-bit) in unsigned decimal format.
- **bud.** - Display a Big-format double (32-bit) number in unsigned decimal format.

- **nu.** - Display a Normal-format number in unsigned decimal format.
- **bu.** - Display a Big-format number in unsigned decimal format.

Formatting

Because ***myforth-arduino*** provides pictured numeric output, custom character display in various formats can be easily added to an application.

For example, ***myforth-arduino*** provides **d.** to display double (32-bit) numbers but there are no corresponding NOKIA commands to do this. If needed, this capablity can be readily added using examples provided in the ***nokia.fs*** source file.

Because most applications require numbers to be displayed with leading zeroes suppressed, the above numeric display words do this. For hex

display, the zero-suppression routine just substitutes a blank for leading zeroes. Thus, values such as 0xab may be displayed as two right-justified characters. Normally, this is what is needed to assure that numbers align. But, the leading blanks may force numeric output off the edge of a line or to the top of a screen, especially with Big-format numbers.

The initialization display for setting the contrast shows a Big-format two-digit hexadecimal value displayed with leading zero suppression.

There are several numeric display protections built into the above commands. Trying to display a number with no number on the stack results in no action, not a "stack underflow" warning. Also, when displaying a zero value, a single zero is displayed instead of all blanks. The zero value in hexadecimal is output as a zero with three leading blanks. If this is not the behavior wanted, a combination of NOKIA (see ***nokia.fs***) and ***myforth-arduino*** words can readily provide what is needed in a custom command.

Fill, Erase

To fill a row with a byte pattern, put the byte on the stack, set the row and execute the **rfill** command. For example, to specify an alternating pattern of bits, enter: **hex 0xy aa rfill** .

To erase a row, you can use **rfill** with a byte pattern of 0x00. For convenience, the command **/row** fills the current row with 0x00. To erase all rows, use the **erase** command. This commands fills all of display memory with zeroes. Do not confuse it with the **blank** command that blanks the display but does not affect the display memory.

To erase all rows and position to xy (0,0), use the the **clear** command. This command is equivalent to: **0xy normal erase** .

Settings

The NOKIA settings are displayed at startup, as shown in Photo 3. These settings can be displayed from a command line or within an application

with the **nset** command. This is often easier than describing individual parameter settings (e.g., instead of **ncon** or **nver** for contrast and version settings).

The current settings displayed with the **nset** command are:

- version - Version of NOKIA software (e.g., Version 1.5).
- contrast - Contrast (e.g., D0).
- option - Option: "cmd option" for command, "demo option" for demo.
- backlight - Backlight PWM value (e.g., bklight 1F).

Use the following commands to display individual settings:

- **nver** - Version of NOKIA software
- **ncon** - Contrast
- **nopt** - Option (e.g., command or demo modes)
- **npwm** - Backlight PWM value

There are several lines available on the option display screen for displaying other values. These can be useful for debug and for application extensions.

Control

There are several commands that control the Nano or provide functions not directly related to the display. These are collected here with the intent that the commands may be expanded to encompass additional "processor only" functions related to the use of unused I/O or a standalone application.

To reset the Nokia display, enter: **~reset**. The "twiddle" prefix indicates that the Nokia's reset line is momentarily changed to perform a reset. The reset line is normally held high (3.3 Volts). Performing a reset pulls the reset line low for one millisecond. This function is rarely used and some users may prefer to permanently wire the reset line high (3.3 Volts).

The command **perm** provides a delay of 3 seconds. It was defined for convenience in implementing the demo display but is also available to the user. The value of the delay can be changed in the ***nokia.fs*** source file by changing the "persistence" constant.

A related command, **peek** is also used in the demo and is available to the user. It delays for 3 seconds and then clears the display (i.e., it executes the **clear** command after a delay). Delays can be performed from the

console by putting the desired delay (in milliseconds) on the stack and using the **ms** command. For example: **decimal 1000 ms** . Note that, during delays, the terminal does not respond to any commands.

Command Summary

Command	Description
BACKLIGHT	**BACKLIGHT**
ba	**backlight alert - rapidly flash backlight**
bas	**set backlight from stack - set longer duration baclight alert w parameter on stack**
bh	**backlight high - set the backlight to a high level (0xc0)**
bl	**backlight low - set the backlight to a low level (0x40)**
bm	**backlight medium - set the backlight to a medium level (0x7f**
bn	**backlight on - set the backlight to the maximum (0xff)**
bf	**backlight off - turn off backlight (set to 0x00)**
bs	**set backlight parameter - set backlight from stack parameter decimal 20 bs)**
pwm	**set backlight pwm value - set backlight from stack parameter 7f pwm). This directly stuffs the fast pwm register.**
save-backlight	**save current backlight setting to eeprom - saved value is res startup**

set-backlight	set backlight from eeprom
@ebl	put backlight eeprom setting on stack. Display this with "." o commands.
!ebl	save stack value to backlight eeprom. This backlight setting at startup.

Command Summary (Cont.)

Command	Description
CONTRAST	CONTRAST
!ncon	set Nokia contrast - sends a new contrast value to the Nokia d the stack
ncon	display the eeprom contrast setting
@econ	put the current eeprom contrast setting on the stack
!econ	set the eeprom contrast setting from the stack
/econ	store the default contrast setting in eeprom - to take effect, th must be reset
set-contrast	set the contrast from eeprom
new-contrast	sets the eeprom contrast value to 0xff to simulate a newly-dow NOKIA program - use with the init-contrast command to allow setting of the contrast (e.g., new-contrast init-contrast)
MODES	MODES

all-on	turns on all Nokia display pixels
blank	blanks the display - Note: does not affect display memory (res display memory with the normal command)
inverse	displays the screen in the inverse contrast mode (dark backgr
normal	displays the screen in the normal contrast mode (light backgr
POSITIONING	POSITIONING
!x	sets the "x" position to the value on the stack. Valid positions from 0 to 83.
!y	sets the "y" position to the value on the stack. Valid positions from 0 to 5. This can also be used as a "row" positioning comm
!xy	sets the "x" and "y" position to the values on the stack. Note t position is put on the stack first, followed by the "y" (row) pos Thus, 0 1 !xy positions to the beginning of the second row (the is 0).
0xy	sets the xy position to the start of the display (upper left corn equivalent to 0 0 xy.

Command Summary (Cont.)

Command	Description
TEXT	TEXT
nchar	Displays a Normal character at the current position using the A stored on the stack. For example, to display an "G" at position enter: "0xy hex 47 nchar".

nstring	Displays a full screen of normal mode text characters (used in application).
~nstring	Displays a full screen of inverse mode text characters (used in application).
bchar	Displays a Big character at the current position using the ASCI the stack. For example, to display an "W" at position (0,0), ent hex 57 nchar".
bstring	Displays a full screen of Big characters in normal mode (used i demo application).
~bstring	Displays a full screen of Big characters in inverse mode (used demo application). This is the same effect as entering: "bstrin

Command Summary (Cont.)

Command	Description
NUMBERS	NUMBERS
nh.	Displays the number on the stack in Normal characters at the position, in hexadecimal. Leading zeroes are suppressed. For "clear hex 0bc0 nh.".
nu.	Displays the (16-bit) number on the stack in Normal character current postion, as an unsigned number. Leading zeroes are s Examples: "clear decimal 0065535 nu." or "clear hex ffff nu.".
nud.	Displays the (32-bit) double number on the top two stack posi Normal characters, at the current position, as an unsigned nu Leading zeroes are suppressed. Examples: "clear decimal 655 or "clear hex ffff ab nud." Note that, as in normal Forth conver most significant byte of a double number is on the top of the s
bh.	Displays the number on the stack in Big characters at the curr postion, in hexadecimal. Leading zeroes are suppressed and r

	blanks. For exmaple: "clear hex 0bc0 bh." Only one four-chara hexadecimal number can be displayed per row.
bu.	**Displays the (16-bit) number on the stack in Big characters, at current postion, as an unsigned number. Leading zeroes are su Examples: "clear decimal 0065535 nu." or "clear hex 00ffff nu. bit decimal or hex number can be displayed on a line.**
bud.	**Displays the (32-bit) double number on the top two postions o in Big characters, at the current position, as an unsigned numb Leading zeroes are suppressed. Examples: "clear decimal 6553 or "clear hex ffff fff nud." Note that, as in normal Forth conver most significant byte of a double number is on the top of the s**

Command Summary (Cont.)

Command	Description
SETTINGS	**SETTINGS**
nset	**Displays all settings in a single screen. See Photo 3**
nver	**Displays the version setting**
ver	**Puts version number on the stack for later use (e.g., output wi display with ".s")**
cver	**Outputs the version string to the terminal ("console version")**
ncon	**Display the contrast setting (from eeprom)**
nopt	**Displays the startup option. Display is "cmd option" for the co mode option (commands executed from a serial terminal). Disp "demo option" for startup in the demo mode. Enter the demo m executing "go" (escape only via a reset of the Nano).**

npwm	**Displays the Pulse Width Modulation (PWM) value (0x00 to 0xf backlight output.**
CONTROL	**CONTROL**
~reset	**Resets the Nokia device.**
perm	**Performs a delay of 3 seconds before accepting the next comm for display persistence).**
peek	**Performs a delay of 3 seconds before accepting the next comm clears the display (used for transient display persistence).**
ms	**Performs a delay based on the value on the stack (e.g., "100 m perform a 100 millisecond delay). Note that the terminal will n to a command until the delay is complete. This function is mos defining display behavior within a *myforth-arduino* definition.**

Revision Summary

Revision	Date	Description
1.5	20Jun16	Expanded command table
1.4	16Jun16	Initial production release

email for support