



599 Menlo Drive, Suite 100
Rocklin, California 95765, USA
Office: (916) 624-8333
Fax: (916) 624-8003

General: info@parallax.com
Technical: support@parallax.com
Web Site: www.parallax.com
Educational: www.stampsinclass.com

NX-1000-24/40 Development Board (#28137)

Development Platform for 24- and 40-pin Microcontrollers

Introduction

The NX-1000-24/40 is a high-quality development and prototyping platform for 24- and 40-pin microcontroller modules. As with the original NX-1000, the NX-1000 24/40 is an excellent platform for BASIC Stamp[®] and Javelin Stamp[™] project development.

In addition to the 40-pin microcontroller socket, there are other differences between the NX-1000-24/40 and the original NX-1000 platform. The table below details the features of each.

Feature	NX-1000	NX-1000-24/40
Microcontroller Socket	24-pin	40-pin
Discrete LEDs	16	16
7-Segment LEDs	4	4
Parallel LCD Connection	Yes	Yes
Audio Amplifier (LM386)	Yes	Yes
Piezo Speaker	Yes	Yes
10K Potentiometer	1	2
N.O. Pushbutton (pulled-up)	8	4
DIP Switch (pulled-up)	8	0
High Current Driver (ULN2003)	7 channels	5 channels
Pulse Generator	Yes	Yes
Secondary RS-232 Connection	Yes	No
PCF8574A ¹	No	Yes
DS1621 ²	No	Yes
DS1307 ³	No	Yes
L272M DC Motor Driver	No	Yes
RJ-11 (for iButton [®] connectors ⁴)	No	Yes

¹ PCF8574A circuit includes four LEDs and four pulled-up DIP switches

² DS1621 circuit includes alarm LED circuit

³ DS1307 circuit includes 32.768 kHz crystal, 3v battery-back-up, and square wave output

⁴ Requires "Blue Dot Receptor" (#805-00004, sold separately) for iButton[®] connection

Packing List

Verify that the NX-1000-24/40 package is complete in accordance with the list below:

- NX-1000-24/40 Development Platform
- 12 volt, 1 amp power supply
- 2x16 Parallel LCD
- DS1990A-F3 iButton sample in plastic fob
- Documentation

Note: NX-1000-24/40 demonstration software files may be downloaded from www.parallax.com.

Questions & Answers

Q: Does then NX-1000-24/40 replace the original NX-1000?

A: No, it is simply a complimentary product.

Q. Can I use the NX-1000-24/40 to run the StampWorks experiments?

A. Yes, most StampWorks experiments can be run on the NX-1000-24/40 platform.

Q. I'm really interested in the BS2p/BS2pe series. Which NX-1000 platform should I use?

A. The NX-1000-24/40 comes with a selection of I²C™ components, and a connection for a 1-Wire® interface making it the ideal choice for BS2p24/BS2pe24 and BS2p40 microcontrollers.

Q. I want to experiment with Dallas/Maxim iButton products ... is there anything else I'll need?

A. You'll need a "Blue Dot Receptor" cable. These are available separately (#805-00004).

Q. Can I use the NX-1000-24/40 for Javelin Stamp projects?

A. Yes, the NX-1000-24/40 is an excellent development platform for the Javelin Stamp.

Q. What is the +V terminal, and how do I use it?

A. The +V terminal is the input voltage to the NX-1000 board (usually 12 volts). This can be used to provide a high-voltage common for devices like unipolar stepper motors that are controlled via the ULN2003 Darlington array.

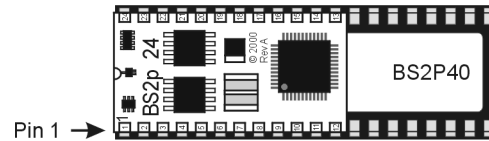


CAUTION: Do not connect the +V terminal directly to any pin of the BASIC Stamp or Javelin Stamp module; doing so will cause damage to or destroy the module. And do not connect +V to any of the +5V or GND terminals as this may damage the voltage regulator of the NX-1000-24/40 board.

The purpose of the +V terminal is to provide +12 volts (input voltage to the NX-1000-24/40) to external devices like relays, incandescent lamps, and unipolar stepper motors that are control via the ULN2003 Darlington array.

Installation of 24-pin Stamp Microcontrollers

When installing 24-pin microcontrollers into the 40-pin socket, take special care to ensure that pin 1 of the BASIC Stamp or Javelin Stamp module aligns with pin 1 of the socket as shown below:



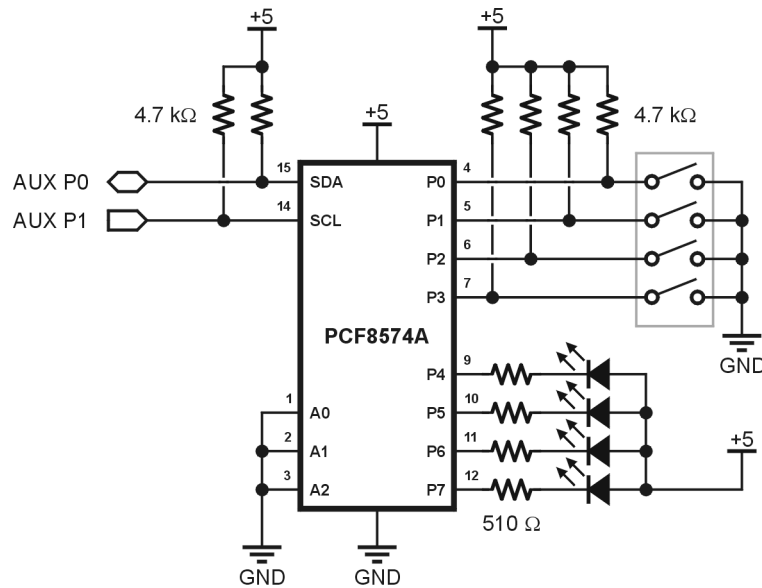
Example Programs

The following pages demonstrate the *new* hardware features of the NX-1000-24/40 development platform (those that differ from the original NX-1000). These programs were written for the BS2p/BS2pe microcontrollers, yet all – with the exception of the 1-Wire[®] example – can be ported to the BS2, BS2e, BS2sx, and the Javelin Stamp, and will run without additional hardware support.

For additional experiments that take advantage of all of the NX-1000-24/40 platform's features (LEDs, Audio Amplifier, ULN2003 Darlington array, etc.), be sure to download our StampWorks documentation.

PCF8574A.BSP

The PCF8574A is an 8-bit, quasi-bidirectional I/O expander that communicates with its host micro through an I²C bus interface. The reference schematic below shows components that are included on the NX-1000-24/40.



```

=====
'
' File..... PCF8574A.BSP
' Purpose.... Demonstrates PCF8574 interfacing using mixed I/O
' Author..... Parallax, Inc. (Copyright 2003 - All Rights Reserved)
' E-mail..... support@parallax.com
' Started....
' Updated.... 22 SEP 2003
'
'   {$STAMP BS2p}
'   {$PBASIC 2.5}
'
=====

' -----[ Program Description ]-----
'
' This program reads and displays the PCF8574A pins P4 - P7 while
' displaying a running counter on PCF8574A pins P0 - P3.
'
' Special Note: When reading inputs while using the PCF8574A in mixed I/O
' mode, you must refresh the output bits during the read. This is easily
' accomplished by ORing the state of the output pins with the DDR value.
'
' I/O Notes:
'
' The input bit is pulled up to Vdd (+5) through 10K. This input is con-
```

```

' nected to Vss (ground) through a N.O. pushbutton switch. The input will
' read 1 when the switch is open, 0 when pressed.
'
' PCF8574A can sink current, but provide almost no source current. Outputs
' for this program are setup as active-low. The tilde (~) in front of
' variables inverts the bits since the PCF8574A uses active-low I/O.

' -----[ I/O Definitions ]-----
SDA                PIN      0                ' SDA on 0; SCL on 1

' -----[ Constants ]-----
DevType            CON      %0111 << 4        ' Device type
DevAddr            CON      %000 << 1          ' address = %000 -> %111
Wr8574             CON      DevType | DevAddr   ' write to PCF8574
Rd8574             CON      Wr8574 | 1         ' read from PCF8574
MixDDR             CON      %00001111         ' 1 = input, 0 = output

#define _P40 = 1                ' change to 0 for BS2p24

' -----[ Variables ]-----
ioBuf              VAR      Byte              ' I/O from/to PCF8574
cntr               VAR      Nib              ' counter

' -----[ Initialization ]-----
Setup:
  DEBUG CLS,
    "PCF8574A Demo"

  #IF _P40 #THEN AUXIO #ENDIF        ' use aux pins on BS2p40

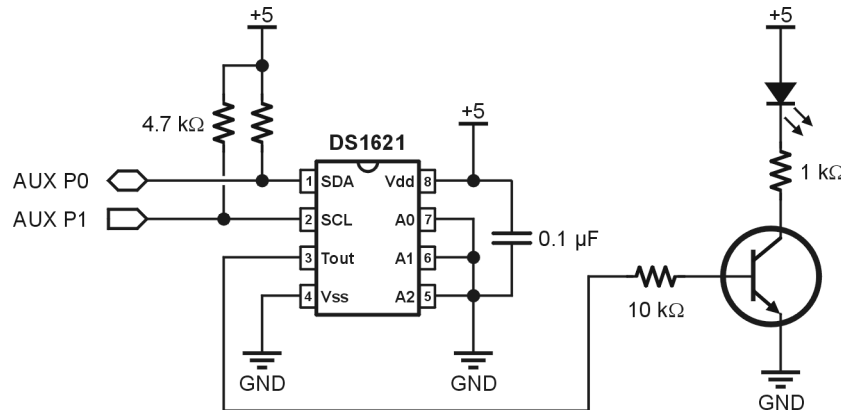
' -----[ Program Code ]-----
Main:
  DO
    FOR cntr = 0 TO 15                ' loop through 4-bit count
      ioBuf = ~cntr << 4 | MixDDR     ' create output byte
      I2COUT SDA, Wr8574, [ioBuf]     ' send LED data / sw mask
      I2CIN SDA, Rd8574, [ioBuf]     ' read switches

      ' report
      DEBUG CRSRXY, 0, 2, "In.... ", BIN4 ~ioBuf.LOWNIB
      DEBUG CRSRXY, 0, 3, "Out... ", BIN4 cntr
      PAUSE 250
    NEXT
  LOOP

```

DS1621.BSP

The DS1621 is digital thermometer and thermostat that communicates with its host micro through an I²C bus interface. The DS1621 includes an alarm output that can be set to activate if the temperature exceeds a stated high threshold, and reset when the temperature falls below a stated low threshold. The reference schematic below shows components that are included on the NX-1000-24/40.



```

' =====
'
' File..... DS1621.BSP
' Purpose... Demonstrates DS1621 interfacing
' Author.... Parallax, Inc. (Copyright 2003 - All Rights Reserved)
' E-mail.... support@parallax.com
' Started...
' Updated... 22 SEP 2003
'
'   {$STAMP BS2p}
'   {$PBASIC 2.5}
' =====
'
' -----[ Program Description ]-----
'
' This program reads and displays the temperature from the DS1621.  Data
' returned is in half-degrees C.  Conversion to Celsius simply drops the
' half degree bit and adjusts to two's complement format if negative.
'
' For conversion to Fahrenheit, the "absolute" temperature relative to
' to -67 degrees it used to simply conversion of negative temperature.
'
' The program also sets the DS1621 levels for thermostat LED control.
'
' -----[ I/O Definitions ]-----
SDA                PIN    0                ' SDA on 0; SCL on 1

```

```

' -----[ Constants ]-----
DevType          CON      %1001 << 4          ' Device type
DevAddr          CON      %000 << 1          ' address = %000 -> %111
Wr1621          CON      DevType | DevAddr    ' write to DS1621
Rd1621          CON      Wr1621 | 1          ' read from DS1621

RdTemp          CON      $AA                ' read temperature
RdCntr          CON      $A8                ' read counter
RdSlope         CON      $A9                ' read slope
StartC          CON      $EE                ' start conversion
StopC           CON      $22                ' stop conversion
AccTH           CON      $A1                ' access high temp limit
AccTL           CON      $A2                ' access low temp limit
AccCfg          CON      $AC                ' access config register

TempHi          CON      25                 ' 26C = ~77F
TempLo          CON      22                 ' 21C = ~72F

DegSym          CON      176                ' degrees symbol

#define _P40 = 1                                     ' change to 0 for BS2p24

' -----[ Variables ]-----
tempIn          VAR      Word               ' raw temp from DS1621
sign            VAR      tempIn.BIT8        ' - sign (after alignment)
halfC          VAR      tempIn.BIT0        ' half-degree C bit
tempC          VAR      Word               ' temp in Celsius
tempF          VAR      Word               ' temp in Fahrenheit

' -----[ Initialization ]-----
Setup:
  #IF _P40 #THEN AUXIO #ENDIF                ' use aux pins on BS2p40
  I2COUT SDA, Wr1621, [AccCfg, %1010]        ' set TOut = active high
  PAUSE 10                                    ' allow EE write
  I2COUT SDA, Wr1621, [StartC]              ' start continuous

Set_Thermostat:
  I2COUT SDA, Wr1621, [AccTH, TempHi]        ' set high threshold
  I2COUT SDA, Wr1621, [AccTL, TempLo]        ' set low threshold

Demo_Screen:
  DEBUG CLS,
    "DS1621 Demo", CR,
    "-----", CR,
    DegSym, "C... ", CR,
    DegSym, "F... "

' -----[ Program Code ]-----

Main:
  DO
    PAUSE 1000                                ' delay between reads

```

```

    GOSUB Get_Temp          ' get current temperature
    DEBUG CRSRXY, 6, 2, SDEC tempC, CLREOL, ' display
        CRSRXY, 6, 3, SDEC tempF, CLREOL
LOOP
END

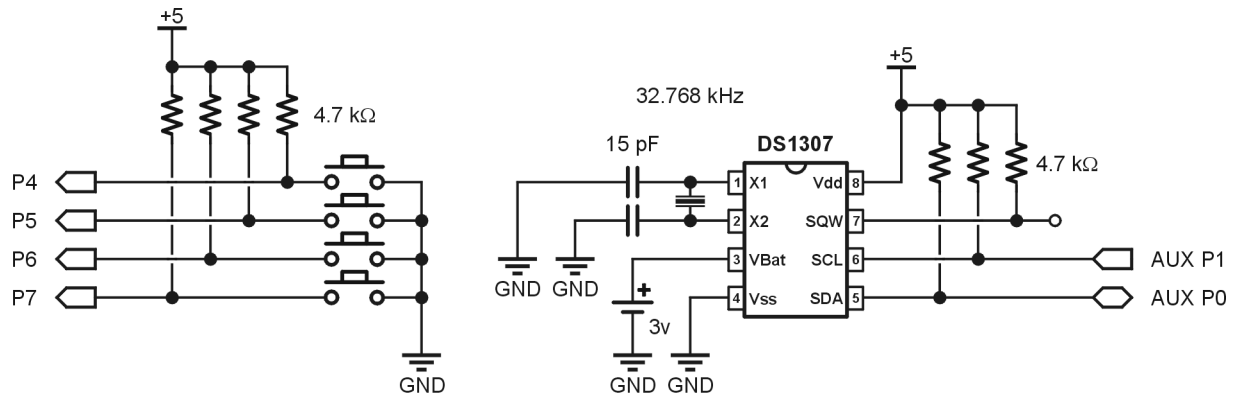
' -----[ Subroutines ]-----

Get_Temp:
#IF _P40 #THEN AUXIO #ENDIF          ' use aux pins on BS2p40
I2COUT SDA, Wr1621, [RdTemp]
I2CIN  SDA, Rd1621, [tempIn.BYTE1, tempIn.BYTE0]
tempIn = tempIn >> 7                ' correct bit alignment
' Celsius
tempC = (tempIn / 2) | ($FF00 * sign)
' Fahrenheit
tempF = (tempIn | ($FF00 * sign)) + 110    ' convert to absolute T
tempF = tempF * 9 / 10 - 67              ' convert to F
RETURN

```

DS1307.BSP

The DS1307 is a low-power, Real-Time-Clock with 56 bytes of NV SRAM that communicates with its host micro through an I²C bus interface. The reference schematic below shows components that are included on the NX-1000-24/40.



```

=====
'
' File..... DS1307.BSP
' Purpose.... Demonstrates DS1307 interfacing and display
' Author..... Parallax, Inc. (Copyright 2003 - All Rights Reserved)
' E-mail..... support@parallax.com
' Started....
' Updated.... 22 SEP 2003
'
'   {$STAMP BS2p}
'   {$PBASIC 2.5}
'
=====

' -----[ Program Description ]-----
'
' This program demonstrates the DS1307 and selective time display modes.

' -----[ I/O Definitions ]-----

SDA           PIN      0           ' SDA on 0; SCL on 1
Mode          PIN      7           ' display mode input
BtnBank      VAR      INB         ' buttons on 4, 5, 6

' -----[ Constants ]-----

Wr1307       CON      %11010000   ' write to DS1307
Rd1307       CON      %11010001   ' read from DS1307

Clock12      CON      1           ' AM/PM display
Clock24      CON      0           ' 24 hour display

```

```

#DEFINE _P40 = 1                                ' change to 0 for BS2p24

' -----[ Variables ]-----
secs          VAR      Byte                    ' DS1307 time registers
mins          VAR      Byte
hrs           VAR      Byte
day           VAR      Byte
date          VAR      Byte
year          VAR      Byte
control       VAR      Byte                    ' SQWV I/O control

buttons       VAR      Nib                    ' debounced button inputs
btnHr         VAR      buttons.BIT2           ' advance hours
btnMn         VAR      buttons.BIT1           ' advance minutes
btnSc         VAR      buttons.BIT0           ' advance seconds

idx           VAR      Nib                    ' loop counter
addr          VAR      Byte                    ' DS1307 RAM address
ioByte        VAR      Byte                    ' data to/from DS1307

oldSecs       VAR      Byte                    ' last seconds read

' -----[ EEPROM Data ]-----
ClockSet      DATA    $FF                    ' $FF = not set

' -----[ Initialization ]-----
Setup:
  READ ClockSet, ioByte                        ' check preset val
  IF (ioByte = $FF) THEN                       ' if $FF we should preset
    #IF _P40 #THEN AUXIO #ENDIF                ' use aux pins on BS2p40
    hrs = $06                                  ' 6 AM
    I2COUT SDA, Wr1307, 0, [STR secs\7]        ' initialize all RTC regs
    WRITE ClockSet, $00                        ' clock has been preset
  ENDIF

  DEBUG CLS,
    "DS1307 Demo", CR,
    "-----"

  oldSecs = $99                                ' force display update

' -----[ Program Code ]-----
Main:
  DO
    GOSUB Get_Time                             ' get current time
    IF (secs <> oldSecs) THEN                  ' update display
      DEBUG CRSRXY, 0, 2
      #IF _P40 #THEN MAINIO #ENDIF             ' mode on main io bank
      IF (Mode = Clock12) THEN                 ' 12 hour mode
        ioByte = hrs // 12

```

```

    IF (ioByte = 0) THEN ioByte = 12
    DEBUG DEC2 ioByte, ":", DEC2 mins, ":", DEC2 secs
    IF (hrs > 11) THEN
        DEBUG " PM"
    ELSE
        DEBUG " AM"
    ENDIF
ELSE
    ' 24 hour mode
    DEBUG DEC2 hrs, ":", DEC2 mins, ":", DEC2 secs,
        CLREOL
ENDIF
oldSecs = secs
ENDIF

GOSUB Scan_Buttons
IF (buttons > 0) THEN
    IF (btnHr) THEN hrs = hrs + 1 // 24      ' update hours
    IF (btnMn) THEN mins = mins + 1 // 60   ' update minutes
    IF (btnSc) THEN secs = secs + 1 // 60   ' update seconds
    GOSUB Set_Time
    oldSecs = 99                            ' force display update
ENDIF

    PAUSE 200                               ' button repeat rate
LOOP

' -----[ Subroutines ]-----
Scan_Buttons:
    #IF _P40 #THEN MAINIO #ENDIF            ' buttons on main io bank
    buttons = %0111                         ' assume pressed
    FOR idx = 1 TO 10
        buttons = buttons & ~BtnBank       ' inputs are active-low
        PAUSE 2
    NEXT
    RETURN

Set_Time:
    secs = (secs / 10 << 4) + (secs // 10) ' decimal to BCD
    mins = (mins / 10 << 4) + (mins // 10)
    hrs = (hrs / 10 << 4) + (hrs // 10)
    #IF _P40 #THEN AUXIO #ENDIF            ' use aux pins on BS2p40
    I2COUT SDA, Wr1307, 0, [secs, mins, hrs] ' update time registers
    RETURN

Get_Time:
    #IF _P40 #THEN AUXIO #ENDIF            ' use aux pins on BS2p40
    I2CIN SDA, Rd1307, 0, [secs, mins, hrs] ' read time registers
    secs = secs.NIB1 * 10 + secs.NIB0      ' BCD to decimal
    mins = mins.NIB1 * 10 + mins.NIB0
    hrs = hrs.NIB1 * 10 + hrs.NIB0
    RETURN

Wr_Reg:
    #IF _P40 #THEN AUXIO #ENDIF            ' use aux pins on BS2p40
    I2COUT SDA, Wr1307, addr, [ioByte]    ' write iobyte to addr

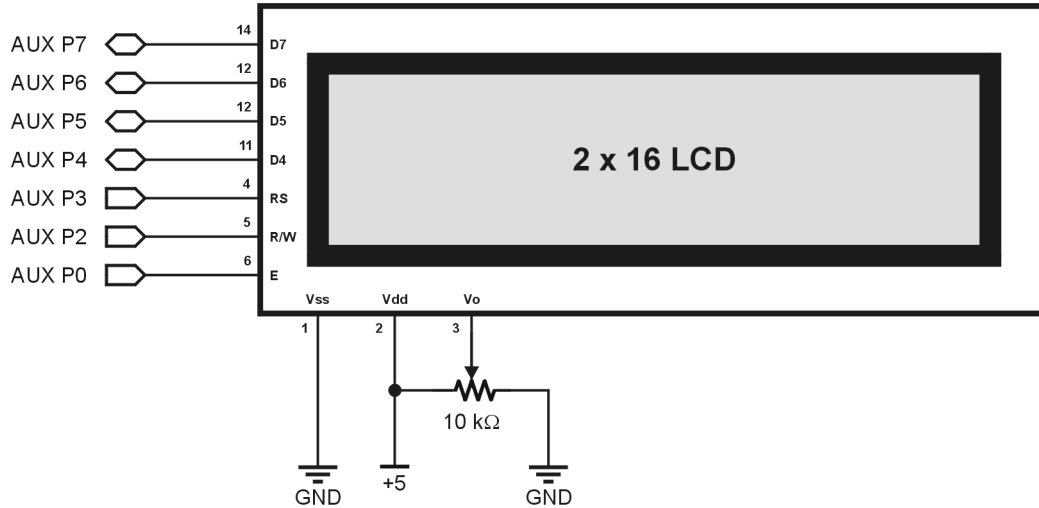
```

```
RETURN

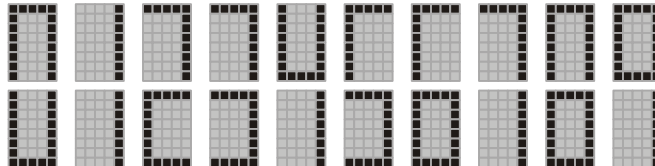
Rd_Reg:
  #IF _P40 #THEN AUXIO #ENDIF          ' use aux pins on BS2p40
  I2CIN SDA, Rd1307, addr, [ioByte]    ' read iobyte from addr
RETURN
```

LCD_DEMO.BSP

This program demonstrates interfacing with a standard parallel LCD, including the use of downloaded custom characters.



The program uses custom characters and code to synthesize 2-line tall digits as shown below.



An additional subroutine uses custom and standard characters to synthesize additional 2-line tall characters that may be useful.

```
' =====  
'  
' File..... LCD_Demo.BSP  
' Purpose... Use LCD custom characters to create tall digits  
' Author.... Parallax  
' E-mail.... support@parallaxinc.com  
' Started...  
' Updated... 22 SEP 2003  
'  
' {$STAMP BS2p}  
' {$PBASIC 2.5}  
'  
' =====
```

```

' -----[ Program Description ]-----
'
' This program demonstrates the creation of tall digits via the use of the
' LCD's custom character set.
'
' Output is to a multi-line LCD. Note that tall character appearance is
' best on LCDs with tight line spacing.

' -----[ I/O Definitions ]-----

E                PIN      0                ' LCD Enable (1 = enabled)
LcdDirs          VAR      DIRB             ' dirs for I/O redirection
LcdBusOut        VAR      OUTB
LcdBusIn         VAR      INB

' -----[ Constants ]-----

LcdCls           CON      $01              ' clear the LCD
LcdHome          CON      $02              ' move cursor home
LcdCrsrL         CON      $10              ' move cursor left
LcdCrsrR         CON      $14              ' move cursor right
LcdDispL         CON      $18              ' shift chars left
LcdDispR         CON      $1C              ' shift chars right

LcdDDRam         CON      $80              ' Display Data RAM control
LcdCGRam         CON      $40              ' Custom character RAM
LcdLine1         CON      $80              ' DDRAM address of line 1
LcdLine2         CON      $C0              ' DDRAM address of line 2
LcdLine3         CON      $94              ' DDRAM address of line 3
LcdLine4         CON      $D4              ' DDRAM address of line 4

#define _P40 = 1                ' change to 0 for BS2p24

' -----[ Variables ]-----

char             VAR      Byte             ' std character sent to LCD
tallChar         VAR      Byte             ' tall character to write
tens             VAR      Nib              ' 0.1 second
secs            VAR      Byte
mins            VAR      Byte
theDigit         VAR      Nib              ' digit value to write
pos             VAR      Byte              ' digit position
idx             VAR      Byte              ' loop counter

' -----[ EEPROM Data ]-----

CC0             DATA     $0F, $09, $09, $09, $09, $09, $09, $09
CC1             DATA     $09, $09, $09, $09, $09, $09, $09, $0F
CC2             DATA     $01, $01, $01, $01, $01, $01, $01, $01
CC3             DATA     $0F, $01, $01, $01, $01, $01, $01, $01
CC4             DATA     $0F, $08, $08, $08, $08, $08, $08, $0F
CC5             DATA     $0F, $01, $01, $01, $01, $01, $01, $0F
CC6             DATA     $0F, $08, $08, $08, $08, $08, $08, $08
CC7             DATA     $0F, $09, $09, $09, $09, $09, $09, $0F

```

```

DigMap          DATA    $01, $22, $34, $35, $12
                DATA    $65, $67, $32, $07, $72

' -----[ Initialization ]-----

Setup:
  #IF _P40 #THEN AUXIO #ENDIF          ' use aux pins on BS2p40

LCD_Init:
  PAUSE 500                            ' let the LCD settle
  LCDCMD E, %00110000 : PAUSE 5        ' 8-bit mode
  LCDCMD E, %00110000 : PAUSE 0
  LCDCMD E, %00110000 : PAUSE 0
  LCDCMD E, %00100000 : PAUSE 0        ' 4-bit mode
  LCDCMD E, %00101000 : PAUSE 0        ' 2-line mode
  LCDCMD E, %00001100 : PAUSE 0        ' no crsr, no blink
  LCDCMD E, %00000110                  ' inc crsr, no disp shift

Download_Chars:                          ' download custom chars
  LCDCMD E, LcdCGRAM                    ' point to CG RAM
  FOR idx = CC0 TO (CC7 + 7)            ' build 8 custom chars
    READ idx, char                      ' get byte from EEPROM
    LCDCMD E, 0, [char]                 ' put into LCD CG RAM
  NEXT

' -----[ Program Code ]-----

Main:
  LCDCMD E, LcdCls                      ' clear the LCD
  PAUSE 250
  LCDCMD E, 0, ["Timer"]                ' simple text on LCD

DO
  FOR idx = 0 TO 1                      ' display minutes digits
    pos = idx + 9
    theDigit = mins DIG (1 - idx)
    GOSUB Write_Tall_Digit
  NEXT

  pos = 11                              ' display cursor
  tallChar = ":"
  GOSUB Write_Tall_Char

  FOR idx = 0 TO 1                      ' display seconds digits
    pos = idx + 12
    theDigit = secs DIG (1 - idx)
    GOSUB Write_Tall_Digit
  NEXT

  pos = 14                              ' display cursor
  tallChar = "."
  GOSUB Write_Tall_Char

  pos = 15
  theDigit = tens
  GOSUB Write_Tall_Digit

  tens = tens + 1 // 10                 ' update tenths

```

```

IF (tens = 0) THEN
    secs = secs + 1 // 60           ' update seconds (if needed)
    IF (secs = 0) THEN
        mins = mins + 1 // 60     ' update minutes (if needed)
    ENDIF
ENDIF
PAUSE 65                          ' pad for 0.1 sec loop
LOOP
END

' -----[ Subroutines ]-----

' Writes a single tall digit
' -- put position (top) in 'pos'
' -- put digit (0 - 9) to print in 'theDigit'

Write_Tall_Digit:
IF (pos < 84) THEN                 ' must be lines 1 - 3
    LCDCMD E, (LcdDDRam + pos)     ' move to top half pos
    READ (DigMap + theDigit), char ' get tall char map
    char = char >> 4               ' extract top char code
    LCDOUT E, 0, [char]           ' write it
    IF (pos < 40) THEN            ' calc bottom half pos
        pos = pos + 64
    ELSE
        pos = pos - 44
    ENDIF
    LCDCMD E, (LcdDDRam + pos)     ' move to bottom half pos
    READ (DigMap + theDigit), char ' get tall char map
    char = char & $0F             ' extract bottom char code
    LCDOUT E, 0, [char]           ' write it
ENDIF
RETURN

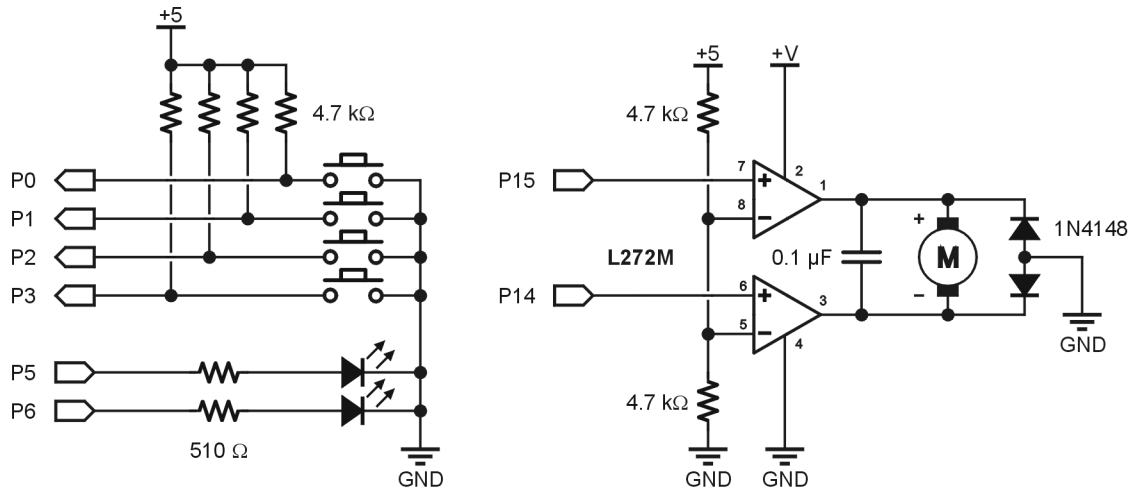
' Writes a single character that is synthesized from
' standard and/or custom characters
' -- put position (top) in 'pos'
' -- pass character in 'tallChar'

Write_Tall_Char:
IF (pos < 84) THEN                 ' must be lines 1 - 3
    char = 99
    LOOKDOWN tallChar, [".:AP"], char
    IF (char < 5) THEN             ' tallChar is valid
        LOOKUP char, [".:",0,0], char ' get top character
        LCDOUT E, (LcdDDRam + pos), [char] ' write it
        IF (pos < 40) THEN        ' calc bottom pos
            pos = pos + 64
        ELSE
            pos = pos - 44
        ENDIF
        LOOKDOWN tallChar, [".:AP"], char
        LOOKUP char, [".:",0,6], char ' get bottom character
        LCDOUT E, (LcdDDRam + pos), [char] ' write it
    ENDIF
ENDIF
RETURN

```

L272.BSP

The L272 is a high-power dual operational amplifier that, in this application, is configured as a bi-directional DC motor driver. The BS2p monitors input buttons that control movement, direction, and speed. Note that the BS2p is used to synthesize the PWM waveform used to control motor speed. The reference schematic below shows components that are included on the NX-1000-24/40.



```

=====
'
' File..... L272.BSP
' Purpose.... Simple DC motor control using the L272M
' Author..... Parallax, Inc. (Copyright 2003 - All Rights Reserved)
' E-mail..... support@parallax.com
' Started....
' Updated.... 22 SEP 2003
'
'   {$STAMP BS2p}
'   {$PBASIC 2.5}
'
=====

' -----[ Program Description ]-----

' -----[ I/O Definitions ]-----

MPos          PIN      15
MNeg          PIN      14

LedCcw       PIN      6
LedCw        PIN      5

btns          VAR      INA          ' pins 0 - 3

```

```

' -----[ Constants ]-----
MStop          CON      0          ' motor states
MCw            CON      1
MCcw          CON      2

LoSpeed        CON      1
HiSpeed        CON      40

' -----[ Variables ]-----

state          VAR      Nib        ' motor state
speed1         VAR      Byte       ' PWM output high time
speed2         VAR      Byte       ' PWM output low time

' -----[ Initialization ]-----

Setup:
  state = MStop

' -----[ Program Code ]-----

Main:
  DO
    ON state GOSUB Motor_Stop, Motor_CW, Motor_CCW
    GOSUB Check_Buttons
  LOOP

' -----[ Subroutines ]-----

Check_Buttons:
  SELECT btns
  CASE %0011
    state = MStop
    LOW LedCw
    LOW LedCcw
    DO : LOOP UNTIL (btns = %1111)      ' force button release
    PAUSE 50

  CASE %0111
    state = MCcw
    LOW LedCw
    HIGH LedCcw

  CASE %1011
    state = MCw
    HIGH LedCw
    LOW LedCcw

  CASE %1101
    speed1 = speed1 + 1 MAX HiSpeed    ' increase speed
    speed2 = speed2 - 1 MIN LoSpeed

  CASE %1110
    speed1 = speed1 - 1 MIN LoSpeed    ' decrease speed
    speed2 = speed2 + 1 MAX HiSpeed

```

```

ENDSELECT
RETURN

Motor_Stop:                                ' stop motor
  LOW MPos
  LOW MNeg
  speed1 = (LoSpeed + HiSpeed) / 2        ' reset default speed
  speed2 = speed1
  RETURN

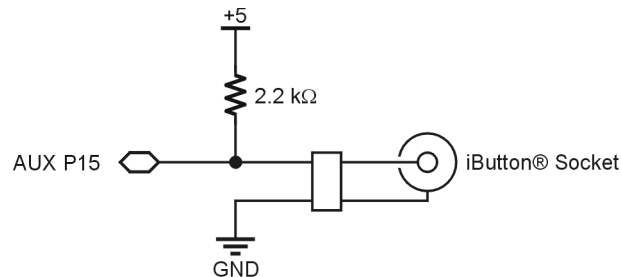
Motor_CW:                                  ' turn clockwise
  HIGH MNeg
  LOW MPos
  PAUSE speed1
  HIGH MPos
  PAUSE speed2
  RETURN

Motor_CCW:                                 ' turn counter-clockwise
  HIGH MPos
  LOW MNeg
  PAUSE speed1
  HIGH MNeg
  PAUSE speed2
  RETURN

```

1W_SECURITY.BSP

This program reads the serial number from a DS1990 iButton and compares it to a list of known valid devices. Note that it requires an iButton probe such as the DS1402 "Blue Dot Receptor" (#805-00004) or similar adapter.



Note: The GND connection is built into the board – the only connection required on the NX-1000-24/40 board is the between the P15 terminal and the iButton terminal.

Note: A program called 1-WIRE_ID.BSP is available to read the serial number from an iButton device.

```
' =====
'
' File..... 1W_SECURITY.BSP
' Purpose.... Simple security system using DS1990A
' Author..... Parallax, Inc. (Copyright 2003 - All Rights Reserved)
' E-mail..... support@parallax.com
' Started....
' Updated.... 22 SEP 2003
'
'   {$STAMP BS2p}
'   {$PBASIC 2.5}
'
' =====
'
' -----[ Program Description ]-----
'
' This program prompts the user to attach a Dallas 1-Wire ID button, reads
' the device serial number and compares it to entries in a table of valid
' codes.  If a match is found, access is granted, otherwise denied.
'
' -----[ I/O Definitions ]-----
'
OwPpin          PIN      15                ' 1-Wire device pin
'
' -----[ Constants ]-----
'
OwFeRst        CON      %0001                ' Front-End Reset
OwBeRst        CON      %0010                ' Back-End Reset
OwBitMode      CON      %0100
```

```

OwHighSpd      CON      %1000
ReadROM        CON      $33          ' read ID, serial num, CRC
SearchROM      CON      $F0          ' search

NoDevice       CON      %11          ' no device present

Yes            CON      1
No            CON      0

#DEFINE _P40 = 1          ' change to 0 for BS2p24

' -----[ Variables ]-----
idx            VAR      Byte          ' loop counter
romData       VAR      Byte(8)       ' ROM data from OW device
devCheck      VAR      Nib           ' device check return code

tries         VAR      Nib           ' scan & id attempts
idNum        VAR      Byte           ' serial num in EE
maxSN        VAR      Byte           ' final SN in EE
eePntr       VAR      Byte           ' pointer to bytes in EE
eeVal        VAR      Byte           ' value from EE
valid        VAR      Bit

' -----[ EEPROM Data ]-----
LastID       DATA     4             ' first ID is 0

ID0          DATA     $01, $63, $08, $B2, $09, $00, $00, $81
ID1          DATA     $01, $84, $BD, $B1, $09, $00, $00, $38
ID2          DATA     $01, $E5, $F8, $B1, $09, $00, $00, $F7
ID3          DATA     $01, $5D, $F7, $B1, $09, $00, $00, $3E
ID4          DATA     $01, $8B, $3B, $55, $09, $00, $00, $0C

' -----[ Program Code ]-----
Main:
DO
  DEBUG CLS, "Attach ID Button"
DO
  GOSUB Device_Check          ' look for device
  LOOP UNTIL (devCheck <> NoDevice)

  #IF _P40 #THEN AUXIO #ENDIF ' use aux pins on BS2p40
  OWOUT OWpin, OwFeRst + OwBeRst, [ReadROM] ' reset iButton
  PAUSE 50
  OWOUT OWpin, OwFeRst, [ReadROM]          ' send Read ROM command
  OWIN  OWpin, OwBeRst, [STR romData\8]    ' read serial number & CRC

  GOSUB Check_ID             ' compare to table entries
  IF (valid) THEN
    DEBUG CLS, "Access granted (ID# ", DEC idNum, ")"
  ELSE
    DEBUG CLS, "Invalid ID. Access denied."
  ENDIF
  PAUSE 1500

```

```

LOOP

END

' -----[ Subroutines ]-----
' This subroutine checks to see if any 1-Wire devices are present on the
' bus.  It does NOT search for ROM codes
'
Device_Check:
devCheck = %00
#IF _P40 #THEN AUXIO #ENDIF          ' use aux pins on BS2p40
OWOUT OWpin, OwFeRst, [SearchROM]    ' reset and start search
OWIN OWpin, OwBitMode, [devCheck.BIT1, devCheck.BIT0]
RETURN

' This subroutine compares the current scanned serial number with table
' entries and sets the variable valid to 1 if match found

Check_ID:
DEBUG CLS, "Checking Device."
PAUSE 500
READ LastID, maxSN                   ' load final entry num
idNum = 0
DO
  valid = Yes                         ' assume code is good
  FOR idx = 0 TO 7                   ' scan bytes in SN
    eePntr = ID0 + (idNum * 8) + idx
    READ eePntr, eeVal               ' get SN byte from EE
    IF (romData(idx) <> eeVal) THEN  ' compare
      valid = No                     ' mark as invalid
      idx = 7                        ' terminate scan loop
    ENDIF
  NEXT
  IF (valid) THEN EXIT               ' found ID
  idNum = idNum + 1                 ' try next ID
LOOP UNTIL (idNum > maxSN)
RETURN

```