HUMTE Demo Project

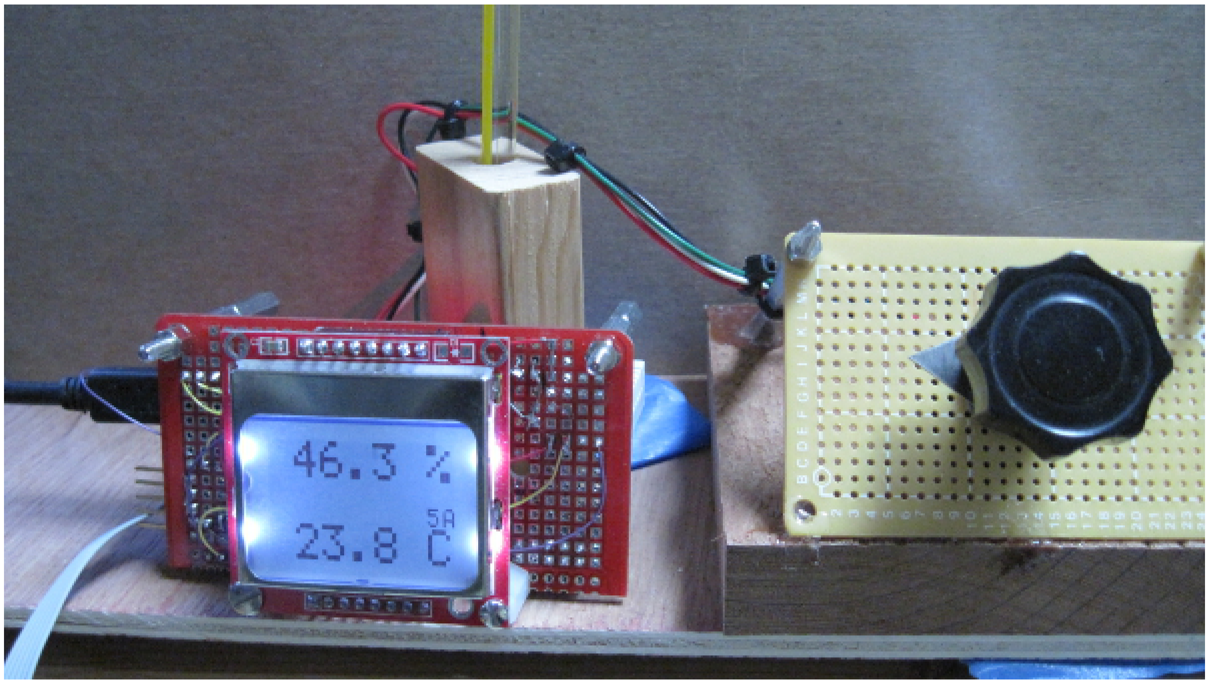

Photo 1. HUMTE Project Development Components

This document describes a demo application using the Nokia 5110 graphics LCD, an incremental encoder and a DHT22 humidity and temperature measurement module. The purpose of this document is to show how three different components with drivers for ***myforth-arduino*** can be integrated to create a small application.

In the following, the demo project is called "HUMTE" (HUMidity TEmperature). Photo 1, above, shows the HUMTE project development assembly using the Nokia development board, an incremental encoder (with a stand and a knob), and the DHT22 humidity and temperature module.

As shown in Photo 1, the demo displays the humidity and temperature using Big characters. The display updates every five seconds as indicated by flashing the display backlight. The encoder is used to determine whether temperature is displayed in Celsius or Fahrenheit degrees.

(BLANK PAGE)

Contents

•

Figures

Introduction

Overview

The HUMTE project uses three components, a Nokia display module, an incremental encoder and a DHT22 humidity and temperature sensor. The project primarily features the DHT22 device and shows a simple application that can be built with the above components. Its value is not so much as a project as it is a demonstration of how various ***myforth-arduino*** drivers can be integrated to build an application.

Also featured is the use Timer 2, operating in the background, to control two different display rates, one for the DHT22 (every 5 seconds) and one for the incremental encoder (every 100 milliseconds).

The timer updates a flag every 100 milliseconds using a Timer 2 driver modified for operation at a 16 MHz clock (see **t2-ticks16.fs**). Alternative timing code shows how to pace display rates without using Timer 2.

Because the DHT22 sensor only uses one I/O line (PB0), the application can use additional I/O for an extended application requiring temperature and humidity readings. The I/O usage for the HUMTE project is shown in Schematic 1.

Introduction (Cont.)

Software

Source code and documentation for the HUMTE project can be found at the ***myforth-rjn*** archive.

The archive provides additional implementation details related to the demo, including the details of the DHT22 protocol, driver coding and error detection. The source is provided to assist in building similar projects requiring temperature and humidity.

The code for the project resides in the Nano module on the Nokia board and was developed with Charley Shattuck's ***myforth-arduino*** system. This is available at the ***myforth-arduino*** archive. It provides useful features of Forth, such as looping constructs and a stack, to be used when they make sense.

However, ***myforth-arduino*** is perhaps most effective when used as a macro-assembler that takes advantage of the versatile instruction set of the Atmel 328p processor. The DHT22 driver (**dht22.fs**), is described in detail to illustrate the use of mixed Forth and assembler instructions.

The demo compiles to a relatively small image, taking approximately 9K bytes of Flash memory, including the standalone command interpreter. The final application also includes a number of utilities, such as math routines, that are generally useful in standalone applications but are mostly unused in the demo.

Introduction (Cont.)

Development

The demo was developed interactively using the standalone interpreter. This development method is described more fully in the NOKIA Manual (see the NOKIA directory in the ***myforth-rjn*** archive).

As noted in the source for Timer 2, (**t2-ticks16.fs**), the timer can conflict with the operation of the standalone interpreter. The application code includes an alternative timing based on a fixed 1 millisecond delay in the main loop (see the Software section below). This alternate code was used during development when the standalone interpreter was needed. The Timer 2 code was installed and verified after the application was nearly complete.

Although the DHT22 driver provides code to do timing with either Timer 2 or main loop timing, there is no advantage in using Timer 2 other than to demonstrate how it might be used in an application where timing is performed in the background.

Future

The author has designed a printed circuit board for the display module, including a DHT22 connection footprint and pullup resistor. The board will soon be sent for fabrication. Contact the author (see last page of this document) if you are interested in purchasing a board or semi-kit.

Quick Start

This section describes the basic demo setup and startup.

First, ensure that the option switch on the Nokia board is in the "APPLICATION" position, per the notation in Schematic 1).

For a new project board, download the latest HUMTE software to the Nano processor on the Nokia board.

After downloading the demo, set the optimal display contrast the first time the demo starts up. The Nokia's on-screen instructions explain how to do this. This process is also described in more detail in the NOKIA Manual.

Assuming that the demo software has been loaded and the contrast has been set, the demo should show the humidity and temperature in Big Nokia characters (see Photo 1). The backlight flashes approximately every 5 seconds whenever the temperature and humidity values are updated.

Observe that there is a two-digit hexadecimal number displayed in Normal characters located just above the Big "C" or "F" characters on the temperature line. This number is a readout of the last byte of the incremental encoder's 32-bit accumulator. This value determines the display mode: either Celsius or Fahrenheit. Values between 0x00 and 0x7e produce a display in Celsius; values between 0x7f and 0xff produce a display in Fahrenheit.

The initial encoder value is zero, resulting in a default display in Celsius degrees. Moving the incremental encoder shaft increments or decrements the initial encoder reading. Moving the encoder slightly counter-clockwise results in an encoder value starting with "F" (e.g., "FC"). With this high value, the display, when next updated, shows the temperature in Fahrenheit.

That is it! Although extremely simple, the demo integrates three commonly-used components to demonstrate a complete turnkeyed application.

Software

Overview

The following sections primarily discuss the features new to this application and provide some insight into the coding strategy. Software included from the base ***myforth-arduino*** system are not discussed unless they relate to the application.

Source code is generally discussed in the order that it appears in the **job.fs** file that builds the application.

Display

The Nokia display driver has been updated from the version given in the NOKIA project and the most current display driver is now **nokia_1.fs** .

Changes primarily relate to the numeric display for unsigned numbers: **nu., nud., bu., and bud.** . Many of these convert the complete number using the pictured numeric operator **#s**. This caused a problem with leading zero suppression in some cases. The code was modified to either use a specific number of digits or to use a larger string to represent a zero value. This cures most problems.

One problematic word is **bud.**. It displays a double number in Big characters as an unsigned number. To fit on a single line of the display, the largest number that can be used is decimal 9999999 (0x98967F). The display of a zero was adjusted to seven places, assuming that, in most cases, numbers would be smaller than this or be displayed in Normal characters. If numbers larger than this value are to carry over to a second line, then custom changes to the code may be needed to handle the "zero" case (e.g., by adjusting the zero string).

Software (Cont.)

Display (Cont.)

A word, **bu.1**, was added to display a number up to three digits and one tenth, in Big characters. This is used to display the humidity and temperature.

Some code inefficiencies were cleaned up, mainly by using the **if/** conditional instead of the **if** conditional. This saves some unsightly **drop** words in the code but is not significantly more efficient.

The words **nspace** and **bspace** were also added to output a space to the display in Normal or Big characters, respectively.

Software (Cont.)

Encoder

The incremental encoder driver, **ie_1.fs**, was changed to correct a potential problem with accessing the encoder's 32-bit accumulator. Previously, the upper and lower accumulator values were fetched separately, disabling interrupts for each value. The code was changed to fetch both upper and lower values at the same time after disabling interrupts.

Two new words were also added, **@encoder-upper** and **@encoder-lower**. These access the upper and lower values of the accumulator separately. Each of these words disables interrupts while fetching a 16-bit value.

One of these new words, **@encoder-lower**, is clipped to 8-bits and used as a control value for display of temperature in either Fahrenheit or Celsius. This word is also useful, for example, when using the encoder to cycle through menu items based on the current encoder value (e.g., by generating an index with **/mod**).

Software (Cont.)

Timer 2

Because the Nokia display module uses a 16 MHz Nano board, the existing Timer 2 code designed for an 8 MHz Pro Mini board was modified for 16 MHz operation. The new Timer 2 code, **t2-ticks16.fs**, is mostly untouched except for new register initialization values.

Clock scaling was changed to 1024 to provide a 10 millisecond tick at the higher clock rate. This tick is counted down in the interrupt service routine to provide a flag that sets every 100 milliseconds. Clearing the flag is the responsibility of the application code using the timer.

The timer initialization word, **/timer2**, was changed so that it did not enable interrupts after execution. This should be done in the startup initialization and was moved there (see the **init** word in **main.fs**, as described below).

Software (Cont.)

DHT22 Sensor

A new driver for the DHT22 sensor, **dht22.fs**, was developed for the demo. The notes section of the source code describes the dht22 protocol.

The DHT22 is a one-wire device attached to PB0. The data line is bidirectional and pulled up to 5 Volts with a resistor (see Schematic 1). The DHT22 driver source defines two words to control whether or not the data line is an output or an input. These words, **dht** and **host**, change PB0 to an input and output, respectively.

Because the sensor data must be acquired rapidly, six temporary registers are defined to hold the five byte message from the DHT22. These registers may be used elsewhere if used atomically. After a DHT22 message is received, the registers are unloaded into the top of stack registers and written to variables for use by the display words.

The documentation for the two-byte temperature and humidity values is relatively clear in the datasheet but there seems to be a bit of misinformation on the Web (imagine that!). The two bytes comprise a 16-bit value that represents the temperature or humidity in tenths.

The new display word, **bu.1**, discussed above, displays the 16-bit temperature value by reading out up to four digits and placing a decimal point before the last digit using pictured numeric output (see **Nokia_1.fs**).

Three words are provided to access the humidity and temperature (in Celsius or Fahrenheit). The word **@humidity** fetches the current humidity reading stored in a variable. The word **@ctemp** does the same for the Celsius temperature.

Software (Cont.)

DHT22 Sensor

The word **@ftemp** fetches the Celsius temperature and converts it to Fahrenheit, taking into account that the value represents the temperature in tenths of a degree.

The conversion algorithm is staightforward: add 400, multiply by 9/5 and subtract 400. This works because, at -40 degrees, Celsius and Fahrenheit are equal. This add just removes the bias before multiplication. Normally, the adjustment would require addition of 40, but 400 must be used because the value is in tenths. Once the adjusted value is scaled by 9/5, the 400 bias is subtracted. Here is the definition:

```
 -: @ftemp ( - n)  @ctemp  400 #, |+  9 #, 5 #, */  400 #, negate |+ ;
```

The word **@chek** fetches the checksum byte from the variable holding the latest checksum. If the checksum doesn't equal the sum of the upper and lower bytes of both the humidity and temperature readings, the value is displayed in reverse video to indicate that it is bogus. The word that does this is **?chek** which is used by the display words.

The word **forc?** checks the least significant byte of the incremental encoder to control the display of either a "C" or a "F" as a Big character next to the temperature value. The value 0x7f is used as a pivot: values below this display a "C" character and values above this display an "F" character.

The word **ie>nokia** displays the least significant byte of the incremental encoder on the display.

Software (Cont.)

DHT22 Sensor (Cont.)

Readout of the bit stream for the five DHT22 data bytes is primarily accomplished by detecting signal edges. Each bit is preceded by a "leader" period of about 50 microseconds when the signal from the DHT22 is low.

At the end of the bit leader, the DHT22 data line goes high. For a zero data bit, the signal remains high for about 25 microseconds; for a one bit, the signal remains high for about 75 microseconds.

The word **sample** detects each bit in the datastream. It first waits for the data output line from the DHT22 to go low. This wait is performed by the macro **-edge** which exits as soon as the leader part of the bit sequence is detected. Once in the leader, another word, **+edge**, waits until the signal line goes high.

At this point, the data signal is at the leading edge of the variable-length high part of the bit sequence. Once this high edge is detected, a fixed delay of 45 microseconds is performed and the data output from the DHT22 is then sampled.

Software (Cont.)

DHT22 Sensor (Cont.)

For a zero bit, the data output at sampling will be low because the 45 microsecond delay is longer that the high duration for a zero. For a one bit, the data output will still be high because the the delay is shorter than the high duration for a one bit. The code to implement a sample is:

```
  \ -: blip      PB2 toggle,  1 #, us  PB2 toggle, ; \ debug marker for
scope
  -: blip ;
  -: edly  1 #, us ;  \ edge delay
  :m +edge  edly begin PB0 set? until  ( PB2 high,) m;
  :m -edge  edly begin PB0 clr? until m;  target
  \
  -: sample  -edge  +edge  45 #, us ;
```

Note that **+edge** is defined with code ("PB2 high,") that can be commented out when debugging with an oscilloscope.

Software (Cont.)

DHT22 Sensor (Cont.)

The bits, as they are detected, are accumulated into a temporary register, **tbits**. The word that accumulates the eight bits for each data byte is **@dht-bits**. This is how it is defined:

```
 -: @dht-bits   0 tbits ldi,
    sample  PB0 set? $80 tbits ori, blip blip  \ bit7
    sample  PB0 set? $40 tbits ori, blip       \ bit6
    sample  PB0 set? $20 tbits ori, blip       \ bit5
    sample  PB0 set? $10 tbits ori, blip       \ bit4
    sample  PB0 set? $08 tbits ori, blip       \ bit3
    sample  PB0 set? $04 tbits ori, blip       \ bit2
    sample  PB0 set? $02 tbits ori, blip       \ bit1
    sample  PB0 set? $01 tbits ori, blip       \ bit0
;
```

The temporary register that accumulates bits is set to zero with the initialization code **0 tbits ldi,**. The instruction **PB0 set?** checks to see if the data line is high at the end of the 45 microsecond bit delay. If it is high, a "one" bit is set in **tbits**; otherwise the instruction that sets the bit is skipped, leaving the bit at zero from the initialization code. Bits are accumulated most significant bit first.

Note that **PB0 set?** compiles a **sbic,** (skip if bit is clear) instruction so that a sequence such as **$80 tbits ori,** is skipped for a zero bit. Note that the bit oring sequence compiles to a single instruction so that the **sbic,** instruction can skip over it.

The **blip** word is for oscilloscope debug. When used, it marks the sampling point with a short signal transition on a spare I/O bit (PB2). Two blips are used to flag the most significant bit.

Software (Cont.)

DHT22 Sensor (Cont.)

The word **@dht** primarily moves the bits accumulated in **tbits** to temporary registers as each message byte is completed. It is defined as follows:

```
: @dht
   host PB0 low,  ( PB2 low,)  25 #, ms   ( PB2 high,)  dht
   +edge               \ wake, high for ~30 us
   45 #, us PB0 set? ; \ exit if no response
   -edge +edge         \ response: low then high for ~80 us
\
\ after 50 us bit leader, a zero is ~26 us high; a one is ~80 usec high
\
\ --- first byte (humidity msb)
   @dht-bits   tbits hmsb mov,
\
\ --- second byte (humidity lsb)
   @dht-bits   tbits hlsb mov,
\
\ --- third byte (temperature msb)
   @dht-bits   tbits tmsb mov,
\
\ --- fourth byte (temperature lsb)
   @dht-bits   tbits tlsb mov,
\
\ --- fifth byte (checksum of previous four bytes)
   @dht-bits   tbits chek mov,
\
   host PB0 high,  ;  \ enter "idle" state
```

Software (Cont.)

DHT22 Sensor (Cont.)

The word definition starts with the code sequence: **host PB0 low, (PB2 low,) 25 #, ms (PB2 high,) dht**. This configures the data line (PB0) as an output from the Nano with the **host** word. Then, the data line is held low for 25 milliseconds to signal the DHT22 to start a new data cycle. When this signal is complete, the data line is turned around with **dht** so that the Nano can receive the output from the DHT22.

The first **+edge** detects the high response from the DHT22 which signals it is awake and not pulling the data line low. This high lasts about 25 microseconds.

The sequence **45 #, us** performs a 45 microsecond delay from the leading edge of the "wake" signal, which should position it in the middle of the low part of the "response" signal. The **PB0 set?** checks to see that the response is low and the DHT22 is active. If the response is high, then the DHT22 is assumed to be inactive or busy and the **;** is executed to exit from **@dht22**. This results in a inverse output on the display, indicating a read error.

If there is no read error, the **;** is not executed and the definition proceeds to the **-edge +edge** sequence. This skips over the remaining part of the "response" sequence. The response sequence consists of (approximately) a 75 microsecond low signal followed by a 75 microsecond high signal. At the end of the response sequence, data bits start with the low leader for the first bit of the first humidity byte.

Software (Cont.)

DHT22 Sensor (Cont.)

The sequence **@dht-bits tbits hmsb mov,** accumulates a data byte and then moves the bit accumulator register **tbits** to a temporary register. In this case the temporary register is **hmsb**, the most significant byte of the two-byte humidity reading.

In a similar fashion, all five data bytes are accumulated in **tbits** and then moved to temporary registers.

When all data bytes have been accumulated, the PB0 line from the Nano is set to an output with a high state. This is the normal "idle" state. In this state, the DHT22 draws minimal current while waiting for the next data query.

The query rate is controlled elsewhere (see below). The datasheet gives a typical value of 2 seconds. Tests with the DHT22 used in development showed that the results were reliable even when the sensor was sampled at a rate of 300 milliseconds. Failure was persistent at 100 millisconds -- this was how the error code in **@dh22** was tested.

The word **.dht** copies the temporary registers used by **@dht** to variables. These variables hold the latest reading of humidity and temperature. The word **?forc**, discussed above, uses the current variables to update the display.

Software (Cont.)

DHT22 Sensor (Cont.)

The word **?dht** controls the timing of the display updates. There are two versions of this word. One verision times display updates with Timer 2. Another version does not use interrupts, deriving its timing by counting the number of times the main loop is executed. It has a one millisecond delay to set the minimum resolution for timing.

The version of **?dht** that uses Timer 2 uses a flag variable, **'tflag**, that is set every 100 milliseconds by the Timer 2 interrupt service routine. Each time the flag is found set, a timing variable, **'5sec** is incremented. This variable is checked for a value of 50 to trigger a five second display update for humidity and temperature.

The timing variable is reset for the next timing period before calling **.dht** to update the display. Note that the interrupts are disabled during a display update so that they (i.e., IE and T2 interrupts) do not interfere with the critical timing of **@dht**.

Software (Cont.)

DHT22 Sensor (Cont.)

To produce a smooth display of the incremental encoder value, it is updated every 100 milliseconds. This timing is also controlled by the **'tflag** variable that is set in the interrupt service routine. Here is the pertinent code for timer-based display updates:

```
 cpuHERE constant '5sec         2 cpuALLOT  \ DHT22 5 second counter
 cpuHERE constant 'ie-timer     2 cpuALLOT  \ timer for IE display
 \
 -: /dht   0 #, '5sec #, |! ;
 -: /ie-timer  0 #, 'ie-timer #, |! ;
 -: /timers  /dht  /ie-timer ;
\
 -: ?dht  \ executes DHT22 read every 5 seconds (timer-based)
   'tflag #, cli, |c@  dup if/     \ $ff if timeout, resets flag if t/o
       0 #, 'tflag #, |c!  ( PB2 toggle,)  \ reset every 100 ms
       '5sec #, |@  1 #, + '5sec #, |!     \ bump 5 second accumulator
    then  sei,  ( -  ?)
 \
    if/ ie>nokia then  \ update encoder based on 100 ms 'tflag changes
 \
    '5sec #, |@  $32 #, negate |+ if/  ;  \ not equal to 50, exit
    then  ( PB2 toggle,) /dht  cli, .dht sei, ;   \ display if equal to
50
```

In the above code, note that the check for a five second display upate happens after managing the **'tflag** variable and checking for an encoder update. The reason for this is that the five second check can directly exit if an update is not needed (i.e., using the ";" after "if/"). If this logic was put before the encoder update, for example, the routine would exit before checking for an encoder update.

Software (Cont.)

DHT22 Sensor (Cont.)

For timing based on main loop delays, the code is as follows:

```
 -: ?display-dht
    '5sec #, |@  1 #+ dup '5sec #, |!
    5000 #, negate |+ if/ ; then ( PB2 toggle,) /dht  .dht ; \ DHT22
display
 -: ?display-ie
    'ie-timer #, |@ 1 #+ dup 'ie-timer #, |!
    100  #, negate |+ if/ ; then /ie-timer  ie>nokia ;
 -: ?dht   ?display-ie  ?display-dht ;
```

This code uses **'5sec** to count the number of times through the main loop. It is bumped up every time through the loop and, when it reaches 5000 (milliseconds), it uses **.dht** to update the humidity and temperature display. The word **/dht** resets **'5sec** to zero whenever a display update occurs. The five second timing is performed by **?display-dht**.

Timing for the incremental encoder is performed by **?display-ie** using code that is similar to that used for **?display-dht**. The word **?dht** consolidates encoder and humidity/temperature timing within the main loop.

Software (Cont.)

DHT22 Sensor (Cont.)

Finally, a note about sensor accuracy. The DHT22 used during development was verified using a laboratory-grade thermometer calibrated in degrees Celsius and Fahrenheit. Readings between the DHT22 and the thermometer were generally in agreement within a half-degee Fahrenheit. The humidity was checked by querying the humidity for the development location, Carmichael, CA, and was found to be in good agreement (typically within a percent or two).

The datasheet for the DHT22 specifies a temperature accuracy of less than half a degree Celsius (about one degree Fahrenheit). The humidity accuracy was specified as plus or minus two percent and is temperature compensated.

Software (Cont.)

Main Loop

The following code implements **go**, the startup word for the turnkeyed demo:

```
 \ Note: use /timer if using interrupt timing; /timers if using main
loop
 \
 : init  /chip  /pwm  /int0  /encoder  ( /timer2)  /timers
         /nokia  /dht  clear  bm  sei, ;
 \
 : go   init  .dht  begin 1 #, ms  ?dht again
```

The initialization word, **init**, initializes all of the application components and enables interrupts. The **/chip** word, initializes the I/O. The I/O initialization is the same as that used for the IE Project but with PB0 set initially as an input (it is used for the DHT22 bidirectional data line).

The **/pwm** word initializes pulse width modulation for the Nokia backlight. The **/int0** word initializes interrupt 0 used by the incremental encoder. The **/encoder** word initializes the incremental encoder registers.

For application timing with Timer 2, the word **/timer2** should be uncommented and the word **/timers** should be commented out. For timing with the main loop, **/timers** should be used, as shown above.

The words **/nokia** and **/dht** initialize the display and the DHT22 senor, respectively. The **clear bm** sequence clears the Nokia display and sets a "medium" backlight. The **sei,** enables interrupts.

Software (Cont.)

Main Loop (Cont.)

The turnkey word, **go**, implements the main loop. Before entering the main loop, the application is initialized and the display is updated with **.dht**. Thereafter, display updates are controlled in the main loop.

The main loop is implemented with the code sequence: **begin 1 #, ms ?dht again**. The **begin ... again** implements an endless loop as required for a turnkeyed application. Within this loop, **1 #, ms** implements a one millisecond delay (see **delay.fs**). The **?dht** word times the display of temperature and humidity and the incremental encoder, as discussed above.

Documentation

This manual is saved in the ***doc*** directory in both HTML and PDF format. The HTML format was 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 ***humte_files*** directory. A separate folder under this directory, ***hr***, provides high-resolution versions of the photos used in the HTML document.

The page numbers apply to the HTML document, which can be printed with the proper page breaks. To see what the printed HTML will look like, use the "print preview" (or similar) option that is availble on most browsers.

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 when used locally (i.e., there is access to the ***humte_files*** directory).

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

Documentation in the source code provides additional information and application usage examples.

Revision Summary

Revision	Date	Description
1.0	24Jul16	Initial release

email for support