NorthTek™ -- Application Note 1002

Emulating SP300X power on characteristics with GEDC-6/DC-4/AHRS-8 NorthTek™ enabled navigation sensors.

Introduction

This application note describes a method of programming the GEDC-6/DC-4/AHRS-8 navigation sensors to emulate the power on characteristics of the SP300X navigation sensors. The SP300X navigation sensors provided a serial output string consisting of the letter "B" followed by a single character from 1 to 8 to indicate the baud rate. This was employed by some navigation sensor users to troubleshoot the serial connection between the navigation sensor and the host platform. Additionally SP300X products, when configured to supply a NMEA result message on a repeating basis, would continue to repeat this output after a power cycle. This note demonstrates how to duplicate this behavior in a GEDC-6/DC-4/AHRS-8 NorthTek™ enabled navigation sensor.

Background

The normal navigation sensor behavior for a GEDC-6/DC-4/AHRS-8 navigation sensor from a power on condition is to initialize and wait for input. As part of the boot process the navigation sensor software checks to see if a user program has been loaded in the EEPROM file system. This will be referred to as a User Bootstrap Program (UBP). If there is a valid program stored there, it is loaded and executed after the navigation sensor initialization is complete. The end user may install any valid NorthTek™ program in the UBP area and it will be executed after reset, cold boot, or resumption from low power mode. This application note takes advantage of that process to emulate the SP300X power on behavior.

The full listing of the program is provided at the end of this document. The actual program file can be found on the Navigation sensor website. The program will be described line by line as the Forth programming language may be unfamiliar to some users. To load this program into a navigation sensor, perform the following steps:

1) Connect the navigation sensor to a PC that has a terminal emulator such as Hyperterminal or TeraTerm.
2) Power up the navigation sensor and start the terminal emulator.
3) Verify communications by hitting return a few times and verify that the "OK" prompt is returned.
4) Configure the terminal emulator to add 5 milliseconds of line delay for each transmitted line.[1]
5) Use the "send file" option to send the program to the navigation sensor. Use only the plain text send functions, the use of a specialized protocol such as xmodem is not required.

[1] On TeraTerm this is found on the setup->serial port menu.

5612 Johnson Lake Rd • Deleon Springs, FL • 32130
800.824.0682 • 386.985.4631
www.sparton.com/ne • www.spartonnavex.com

6) Reset the navigation sensor, either by power off, or sending the “reset” command.
7) Observe that the BX string is printed and that the navigation sensor is outputting the NMEA magnetic heading string at a 500 msec rate.
8) To remove the UBP, type 0x10000 eepageerase.

Detailed description of the User Boot Program.

In this section the UBP will described line by line. The user may wish to have the full program listing available as a reference. In Forth, the native language of NorthTek™, a comment is started with the // operator. Note that a space must follow all operators in Forth, thus the reader will note that all comments (in fact all operators) are surrounded by whitespace[2]. Thus in the program listing all lines that start with // are comments. Note that the comment operator may appear on a line of code and the comment extends to the end of the line.[3] The reader is encouraged to read the program comments.

The first lines of code in the program are:

```
0x10000 userOpen
: put start: userWrite ;  // This makes writing a record easy.
```

The line “0x10000 userOpen” does two things; first it pushes the value 0x10000 on to the Forth parameter stack, then it executes the word “userOpen”. As this file is being transferred to the navigation sensor each line of text is being examined by the Forth interpreter. At this point in the transfer only comments have been encountered and thus discarded by the interpreter. The interpreter is not compiling as yet, so when it encounters this line it parses it and performs the actions immediately. This is the reason that the terminal emulator must be set to 5 millisecond line delay as the interpreter is actually executing the statements for that transmitted line as they arrive. In this case the word “userOpen” is being executed with the value 0x10000 passed in as a parameter on the parameter stack. The userOpen word prepares the non-volatile file system by opening a file at the passed in address.

Next the interpreter encounters the line “: put start: userWrite ;”. This line is handled as follows:

1) The interpreter encounters “:”, this switches the interpreter into compile mode.
2) The compiler reads ahead, parses the word “put” and creates a new entry in the Forth wordlist called “put”.
3) The compiler continues parsing the input, encounters “start:” and compiles that into the the “put” definition. This continues with the word (i.e. function) “userWrite”. At this point the word “put” now has two function calls, “start:” and “userWrite”.
4) The compiler parses “;”. This is a special word that says finish the current definition and return to interpret mode.

[2] Newlines or spaces count as white space, tabs are not used.
[3] Inline comments can be done with “(“ and “)”, surrounded by whitespace.

At the end of this line a new word has been added to the Forth wordlist on the navigation sensor.[4] This word, when executed calls two other words, “start:” and “userWrite”. The “start:” word is a parsing word. It gathers all the text on the line that follows up to a carriage return and makes a string out of it (excluding the carriage return), leaving the string information on the parameter stack. The “userWrite” word expects string information on the stack and writes that string to the currently open file in the non-volatile storage. The “userWrite” word also appends a newline to each line written. The “userWrite” word also accumulates the required information to add the error detection header to the file when it is closed.

Note that the “put” word has only been compiled at this point and has not been executed. To summarize the activity up to this point, all the comments have been discarded, the UBP file area has been opened, and a shorthand word to read from the input and write to the open file has been defined.

The interpreter continues with the next line:

```
put : b4. baud di@ <# # drop char B hold #> type ." \r\n"  ;
```

This line is handled as follows:

1) The interpreter parses the word “put” from the input stream.
2) The “put” word is executed. The “put” word executes its two function calls. First the “start:” word is executed, reading all the text up to the carriage return, the “userWrite” function is then called, writing the line of text into the UBP area.

That’s it. The net effect of this line is that the string “: b4. baud di@ <# # drop char B hold #> type .” \r\n” ;” is written into the UBP area at the current file pointer, that happens to be at 0. Note that nothing is compiled or executed related to the string at this time, it is merely written to UBP. The behavior will be described later when the string is executed.

The interpreter now encounters (ignoring comment lines):

```
put b4.
put : % " $xxHDM\n" count nmea! ;
put " $xxHDM,RPT=0.5\n" count nmea!
```

The interpreter goes through the same actions before, it executes “put”. The “put” word acts as before writing the string “b4.” and the following strings into the UBP.

The interpreter then encounters:

[4] This word is transient and is lost as soon as the navigation sensor is powered down or reset.

```
userClose
```

The interpreter executes this word. The “userClose” word tidies up the file, writes the error detection and length to the file and closes the file. The net effect of sending the file to the navigation sensor is to write a program into the UBP area. One temporary word called “put” is defined to help with this process, otherwise nothing else is done. Note that the contents of the UBP program are roughly as shown here (ignoring the internal header, etc.):

```
: b4. baud di@ <# # drop char B hold #> type ." \r\n" ;
b4.
: % " $xxHDM\n" count nmea! ;
" $xxHDM,RPT=0.5\n" count nmea!
```

Upon reset the navigation sensor examines the UBP and finding that the file header is in order and the error checks pass, loads the UBP program into the interpreter; just as if it had been send by serial port. Line by line this is what the interpreter does:

```
: b4. baud di@ <# # drop char B hold #> type ." \r\n" ;
```

- The interpreter parses “:” and enters compile mode.
- The compiler parses “b4.” from the input and creates a new word definition called “b4.”
- The compiler compiles all the tokens from “baud” to \r\n” into the new word.
- The compiler parses “;”, and finishes the new word definition and returns to interpret mode.

```
b4.
```

- The interpreter parses “b4.”, searches for, finds and then executes the word “b4.”
- The word “b4.” executes as follows:
    - The word “baud” is executed, this is a database definition word and returns the object reference information on the stack.
    - The word “di@” is executed. This word takes the object information from the stack and returns the value of that object. In this case the value returned is the baud rate index (1=1200…8=115200, see Software Interface Users Manual for others).
    - The standard Forth word “<#” is executed. This word prepares a format string for formatted output.
    - The standard Forth word “#” is executed inserting one digit of the value on the stack (the baud index) into the output string[5]. Assume for this example that the baud index is 8.

[5] The output string is built up from right to left.

- The standard Forth word “drop” is executed, as only one digit is needed.[6]
- The standard Forth word “char” is executed[7] pushing the letter “B” on the stack.
- The standard Forth word “hold” is executed, adding the the “B” that is on the stack to the output string.
- The standard Forth word “#>” is executed finishing the output string and leaving a string reference on the stack.[8]
- The standard Forth word “type” is executed, outputting the string that contains “B8”.
- The standard Forth word “.”” is executed outputting the string “\r\n”, where the closing quote is a parsing token for “.””. This outputs a carriage return and newline.
- The word “b4.” exits back to the interpreter.

- The net effect is that string “B8\r\n” is output on the serial port at boot time, as desired.

```
: % " $xxHDM\n" count nmea! ;
```

- The interpreter parses “:” and enters compile mode.
- The compiler generates a new definition for the word “%”, consisting of the contents shown.

This word is not executed at this time but is invoked by the user typing “%” as a command and entering carriage return. The effect of executing “%” is:

- The word “”” is executed. This pushes a pointer to a temporary counted string on the stack.
- The word “count” is executed. This takes the pointer to a counted string and converts it into a string reference (a count and a pointer, expected by “nmea!”).
- The word “nmea!” is executed. It removes the string reference from the stack and passes it to the NMEA command parser as if it had been received on the serial port.
- The NMEA command parser executes the heading command with no repeat option, cancelling repeating NMEA output.

The net effect is that if the user executes the % word, NMEA repeating output is cancelled. The interpreter continues encountering the next two lines:

```
" $xxHDM,RPT=0.5\n" count nmea!
```

The interpreter performs the following actions as a result:

- The word “”” is executed. This parses a string up to the next “, then pushes the address. In this case the

[6] Note the NorthTek™ version of #> does not remove the value from the stack per ANSI standard. This allows formatting of more than one number in the output “picture”.
[7] Actually “char” was executed when b4. was being compiled, at runtime the net effect is that the ‘B’ is pushed on the stack at runtime. If interested the reader is encourage to consult a Forth text regarding compile time versus run time behavior of some words.
[8] The reference is two values, a count and a pointer. This is expected by the word “type”.

string is “$xxHDM,RPT=0.5\n”.

- The word “count” converts the pointer into a reference.
- The word “nmea!” transmits the string to the NMEA parser.
- The NMEA parser executes the command, outputting the magnetic heading at 0.5 second rate.

The interpreter now reaches the end of the file and waits for further input.

In summary the net effect at boot time is to output the “B8” string and then begin outputting the magnetic and true heading at a 0.5 second rate, thus producing the desired result. The NMEA repeating output can be stopped by sending %<cr>.

Complete Listing of the compatibility program.

```
// ************************************************************
// ************************************************************
// Program to enable legacy navigation sensor NMEA behavior.
// This file loads a set of forth words into the EEPROM
// that will execute at bootup.
// Needs svn revision 2.1.1 or later.
// This program does the following:
// 1) Defines a word call b4.  this word prints out a B followed
//    by the baud rate as an index. B4 would be printed
//    if the baud rate is 9600. B8 would be printed if the baud rate is
//    115200.
// 2) Executes the b4. word so it is printed at startup.
// 3) Defines a word %, that will stop NMEA repeats.
// 4) Issues a NMEA command with repeat at startup.
//    In this case the command is to output heading with
//    a repeat rate of 0.5 seconds.
//
// This program is sent to the navigation sensor over the serial port.
// Upon reboot the navigation sensor will output "BX"<CR><LF>
// Then it will echo the NMEA command and then an OK.
// Then the repeating output will occur at the rate specified.
// ************************************************************
// ************************************************************

// ************************************************************
// Open the user space file.
// ************************************************************
0x10000 userOpen
: put start: userWrite ;  // This makes writing a record easy.
// ************************************************************
// this record prints BX
// start a formatted string, format one character
// then drop the rest.
// Insert the B in front, type it out, then type cr,lf.
// ************************************************************
put : b4. baud di@ <# # drop char B hold #> type ." \r\n"  ;
// ************************************************************
// Run the word
// ************************************************************
put b4.
// ************************************************************
// Define the % word that allow nmea repeat suppress.
// Type %<CR> to execute.
// ************************************************************
put : % " $xxHDM\n" count nmea! ;
```

```
// ************************************************************
// Issue the NMEA repeat command at startup
// ************************************************************
put " $xxHDM,RPT=0.5\n" count nmea!

// ************************************************************
// ************************************************************
// Close the user space file
// ************************************************************
userClose
```

Want to know more?

- Check it out here: www.spartonnavex sensor.com