Wavemeter User Manual

Tobias L. Brown-Heft
University of Oregon Atom Optics
Dr. Daniel Steck

July 31, 2013

Contents

1 Overview

The wavemeter is a scanning Michelson interferometer that can be used to determine the wavelength of an unknown laser if the wavelength of a reference laser is known. Useful information is processed by the device and sent to the Rubidium server where the data is analyzed and displayed on a website.

This manual is intended mainly to document the reasoning behind the more important updates of the wavemeter design and programming over the last few years in the hope that future operators will be able to track down and resolve issues efficiently.

1.1 Accuracy

Wavemeter v2.2 has an error of 10ppm or lower when properly calibrated. Data collected using a 780nm reference beam locked using ^{87}Rb saturated absorption spectroscopy was compared to an unlocked but stable 914nm test beam. The lasers were first tested in Hailin Wang's WA-1000 commercial wavemeter, then measured with our wavemeter to compare the results. The WA-1000 was confirmed to be accurate by inputting a thermally stabilized HeNe laser and measuring 632.825nm. This is in good agreement with the calculated wavelength of 632.823nm based on the accepted HeNe vacuum wavelength 632.991nm and an index of refraction of 1.000265931 (air, 22 °C, 632.991nm, 1 Bar, 50% humidity) calculated by Wolfram-Alpha. So the WA-1000 can be trusted to within 0.002nm or so. You can use the *Ciddor* or *Edlén* equations if you wish to calculate index of refraction manually.

The theoretical relative error of the wavemeter is one fringe out of the total number of fringes collected, or about $\frac{1}{5\times10^6} = 2\times10^{-7}$. In order to test the actual absolute accuracy of the wavemeter it will be necessary to feed it two lasers that are both locked to well-known atomic resonances. This is a little excessive, since a relative error of 10^{-5} should be fine for our purposes anyway.

The 780nm Rubidium 87 D_2 ($5^2S_{1/2} \longrightarrow 5^2P_{3/2}$) transition used for testing has a frequency of $2\pi \cdot 384.2304844685$ THz and a linewidth of $2\pi \cdot 6.0666$ MHz according to Table 3 of *Rubidium 87 D Line Data* authored by Daniel Steck. This gives a relative uncertainty of 1.58×10^{-8}, which is an order of magnitude lower than the wavemeter counting uncertainty and is therefore negligible.

WA-1000		Wavemeter v2.2	Error	
Reference (nm)	Test (nm)	Test (nm)	Absolute (nm)	Relative ($\times10^{-6}$)
780.027	913.712	913.702	0.010	10.9
780.028	913.717	913.719	-0.002	-2.19
780.031	913.733	913.729	0.004	4.38
780.031	913.731	913.734	-0.003	-3.28
782.108	913.692	913.6914	0.0006	0.66
782.115	913.708	913.7033	0.0047	5.14
782.116	913.706	913.7024	0.0036	3.94
782.117	913.705	913.7014	0.0036	3.94
782.118	913.703	913.7005	0.0025	2.74
782.120	913.706	913.7020	0.0040	4.38
782.115	913.693	913.6909	0.0021	2.30
782.115	913.683	913.6844	-0.0014	-1.53

2 Operation

2.1 Laser Sources

The wavemeter requires a known reference beam. Atom trapping experiments frequently use laser sources that are locked to atomic transitions, which are typically known to very high accuracy and precision. Any such locked laser will make an excellent reference beam. Input this laser into the wavemeter reference fiber couple. See Section 3.4. The wavelength of the reference beam should be input into both the Ethernut's non-volatile memory by accessing telnet>debug>lcd>reference, and into the wavemeter data website. The test beam can be any wavelength, but must be single mode and reasonably stable over time periods of order 30 seconds. Input the test beam into the wavemeter test beam fiber couple.

Connect an oscilloscope to the Reference and Test Photodiode Out BNCs on the wavemeter faceplate. Use the Manual Direction Control to move the cart and produce a fringe signal. The fringe signals must be almost pure sinusoids to ensure proper operation. Watch out for weak signals due to low power lasers. The lowest power laser which was successfully used as a reference beam was $17.0\mu W$. Anything with less power produces bad data due to the low signal-to-noise ratio. Also watch out for photodiode saturation due to high power lasers. Saturation can be avoided by using an iris or attenuator just before the photodiode. An iris was required for a laser of $143\mu W$.

2.2 Ethernut

On power-up, the Ethernut establishes a LAN connection and telnet communication. The telnet functions can only be accessed by the server at rb.uoregon.edu, so it is necessary to SSH into rb@rb.uoregon.edu, then issue the terminal command “telnet wavemeter”, which initializes communication with the wavemeter.

Currently, the entire wavemeter operation must be initialized via telnet. Power on the wavemeter, wait a few seconds for it to get a connection, then telnet in. Enter the “debug” menu, then type “power” twice to activate the cart translation and data collection.

The Ethernut is programmed to throw away data collected by the pass completed immediately after this command is issued. The LCD screen will update with the bad data to give an indication that at least *some* systems are operating properly, but only full, uninterrupted measurements are sent to the PHP server.

Possibility: Use one of the few remaining Ethernut I/O pins for a simple button on the wavemeter box to activate cart translation. Then no telnet would be required for normal operation. The code would need to modified to include a periodic test of the state of the button. This test could be done in counter.c just after the test to see if the cart has turned around. This would prevent interference with good operation. See Section 4.4 for an explanation of why the timing of this test is important, and Section 5.1 for a list of available Ethernut pins.

3 Interferometer

This is where the important physics happens. Two lasers are aligned through the path of the wavemeter. The alignment procedure can be found on the wavemeter website. Once properly aligned, the two beams are interfered with themselves.

3.1 Production of Fringes

A light wave with wavenumber k and frequency w traveling in the x direction has the complex wavefunction $\Psi(x,t) = Ae^{i(kx-wt)}$. The wave is split evenly into two parts and sent through an interferometer with path lengths

$$P_1 = 2(l_1 + x_c) \quad P_2 = 2(l_2 - x_c), \tag{1}$$

where x_c is the position of the retroreflective cart along its track and l_n is the constant length of the nth leg of the interferometer. When these two beams are recombined and measured at some point z beyond the recombination point they have the wavefunction

$$\Psi(x_c,t) = \frac{A}{2}e^{i(k(P_1+z)-wt)} + \frac{A}{2}e^{i(k(P_2+z)-wt)} \tag{2}$$

$$= \frac{A}{2}e^{i(kz-wt)}[e^{ikP_1} + e^{ikP_2}] \tag{3}$$

But the intensity I of the light wave is given by the absolute square of the wavefunction,

$$I(x_c,t) = |\Psi(x_c,t)|^2 = \Psi^*\Psi \tag{4}$$

$$I(x_c,t) = \frac{A}{2}e^{-i(kz-wt)}[e^{-ikP_1} + e^{-ikP_2}] \times \frac{A}{2}e^{i(kz-wt)}[e^{ikP_1} + e^{ikP_2}] \tag{5}$$

$$I(x_c) = \frac{A^2}{4}(2 + e^{ik(P_1-P_2)} + e^{ik(P_2-P_1)}) \tag{6}$$

$$I(x_c) = \frac{A^2}{2}(1 + \cos[k(P_1 - P_2)]) \tag{7}$$

$$I(x_c) = \frac{A^2}{2}(1 + \cos[k(2(l_1 + l_2) + 4x_c)]) \tag{8}$$

Taking l_1 and l_2 as constant and so only giving an uninteresting phase shift, as well as throwing away the constant amplitude offset gives

$$I(x_c) = \frac{A^2}{2}\cos(4kx_c) \tag{9}$$

This is the observed result when a monochromatic light wave is sent through an interferometer of this design. For a cart at position x_c, the AC coupled power signal measured at point z will be proportional to $\cos(4kx_c)$. The fringes are produced with a *periodicity proportional to the difference in path length* between the two legs of the interferometer.

3.2 Wavelength Comparison

For the path lengths P_1 and P_2 above, the difference in path length is

$$\Delta P = C + 4x, \tag{10}$$

where C is the constant path length difference. This means that there are 4cm of path length difference for every 1cm of cart translation. This has the benefit of allowing us to collect more fringes and increase the accuracy of the final calculation while keeping the interferometer design physically small.

Fringes occur when *changes* in the path length difference occur, or when ΔΔP is an integer multiple of the laser wavelength, $\Delta\Delta P = n\lambda$. But $\Delta\Delta P$ is the same for both lasers due to the action of the retroreflective cart, whereas $\lambda_1 \neq \lambda_2$. This means there will be a different number of fringes over the same ΔΔP for two different lasers. The number of fringes counted for laser 1 is N_1, where $N_1\lambda_1 = \Delta\Delta P$, and likewise $N_2\lambda_2 = \Delta\Delta P$. Then $N_1\lambda_1 = N_2\lambda_2$, and so

$$\lambda_2 = \frac{N_1}{N_2}\lambda_1 \tag{11}$$

Therefore, if we know the ratio of the number of fringes and the wavelength of one laser, we can calculate the wavelength of the second laser.

The longest translation the wavemeter can achieve is $\Delta x \approx 1m$. For a typical laser of 700nm being measured by the wavemeter, the number of fringes the interferometer will produce is $N\lambda = \Delta\Delta P = 4\Delta x$, or

$$N \approx 4 \times 1m/700nm = 5.7 \times 10^6 \tag{12}$$

3.3 Multimode Lasers

The interferometer only works well for lasers operating in single mode. A multimode laser will produce fringe patterns with a lot of noise and beat notes. While perfecting the alignment procedure, I kept seeing ~20Hz beat notes in the fringe signal. I assumed this was due to the system going in and out of alignment since the steel rod isn't completely straight and it was wobbling while rotating. However when a single mode laser was introduced the "wobbling" went away and an almost perfect sinusoid was produced. Trust the alignment procedure on the wavemeter site. It works.

3.4 Fiber Optics

The reference and test beams are delivered to the wavemeter using fiber optic cables. These cables are protected by a plastic conduit we placed in the cable tray that runs above the wavemeter closet into the main lab. Currently there are fibers for 780nm, 689nm, 914nm, and 994nm, and they *should* be labeled. The 780nm fibers go to the Rubidium table, and the rest of them go to the Strontium table. Each fiber coupler has a coupling lens with a particular focal length, and will only work well for fibers which fit snugly into the fiber couple and have their terminus at the coupling lens focal point. But each fiber has a slightly different terminus length, so some fibers won't work in some couplers. *You will need to find a coupler that will work well with the fiber you are using.* Don't try to make plastic washers to fit between the fiber and the coupler. You'll never be able to get a good alignment after removing and replacing a fiber that has been hacked in this way.

Once you have a decent fiber/coupler pair, you'll probably need to realign the wavemeter optics. This is a somewhat tedious process, but it is absolutely essential. The alignment procedure can be found on the wavemeter project site. Once alignment is done for a pair of fibers, it should remain aligned for a long time.

3.5 BNC Cable in the Cable Tray

The more unfortunate aspect of the wavemeter's location is that it has one running back and forth between the wavemeter closet and the main lab. To minimize the relentless jogging around that wavemeter calibration can and will bring, a BNC cable has been installed in the cable tray above the wavemeter closet. Presently one end of the cable resides coiled up in the cable tray above the main office, and the other end dangles behind the door of the closet. It may be worth it to drill a hole in the closet ceiling to run the cable through, but be very careful not to get sheetrock dust on the optics. This cable is very useful for maximizing fiber coupling and ensuring single mode operation. Eventually it may play an important role by allowing operators to see test beam fringe patterns while using the Strontium optical lattice.

3.6 Other Details

The retroreflective cart is translated along two steel rods, one of which rotates, actuating a linear bearing inside the cart. *Do not touch the rods with your hands.* They rust easily. They must be kept coated in a thin film of Krytox GPL 106 Oil. The rod is turned by a bi-directional DC motor which is controlled with two relays. One relay controls on/off, and the other controls cart direction. Currently the motor receives 12 volts from an ATX PSU, but this could be increased for faster operation if desired.

The cart reverses direction when a metal flag on the cart passes into a photodetector at either end of the track and breaks an IR beam. These detectors are positioned at both ends of the track and connect to the wavemeter mainboard.

4 Wavemeter Mainboard

The mainboard ties all the wavemeter systems together, keeps things organized, and performs functions not easily done by the Ethernut. Version 2.0 was produced by interns Keith Fannin and John Smith during the summer of 2011. They created an Osmond drawing and ordered professional PCBs, however there were a few issues with that model. There were thermal pads that didn't actually make contact with the ground plane and missing 5V lines. Wavemeter v2.1 was intended to correct these errors but a PCB was never ordered. Wavemeter v2.2 files have been fully corrected and are totally representative of what is in the wavemeter as of this moment. Again, PCBs were never reordered, but if anyone wants to make another wavemeter, they just need to order v2.2 and it *should* work perfectly the first time.

4.1 Photodiode Input

The fringes produced by the interferometer are detected by homebuilt photodiodes, which have their own project page on the internal site. The wavemeter photodiode input is copypasta from that design so look there for documentation. The mainboard provides power to and receives the signal from the photodiodes at the interferometer output. This rough sinusoidal signal is amplified, 300kHz low-pass filtered to reduce noise, then high-pass filtered to cancel any DC offset at the comparator input.

4.2 Comparator

In order to digitally count the fringes, the outputs of the photodiode amplifiers are sent to separate LM306 comparators, which convert the sinusoids into TTL signals. Adjusting the two 10-turn trimpots allows the user to precisely select the comparator threshold value. The TTL signals are ANDed with a gate signal from the Ethernut and the result is sent to clock0 of the respective 82C54 counter chips. The gate signal is intended to prevent the counters from collecting any fringe data during cart turnaround, since these fringes will be extremely noisy.

4.3 Counters

The wavemeter uses two 82C54 counters to count the number of fringes produced by the interferometer. These chips are capable of clocking at 10MHz, but in this application almost never experience anything above ~150kHz. If the DC motor used to drive the cart is ever provided with a higher operating voltage, it's possible the low-pass filter will need to be adjusted to allow for higher frequencies.

The 82C54 was originally designed to be used in Intel computers and is quite sophisticated as counters go, and was chosen mainly for its large storage capacity. As determined in Section 3.3, the counters will need to regularly deal with numbers around 5 million, or $\sim 2^{22}$. So we need at least 22 bits of counting capacity, which would otherwise require the cascading of three or four independent standard 8-bit counters per laser, depending on how safe one wanted to be.

Each chip contains three internal counters. Only two are used here. Each internal counter is a two-byte down-counter, which provides 16 bits of counting capacity. Two cascaded internal counters therefore provide 32 bits of counting capacity, which is about 1000x more than we need. Each internal counter has an associated one-byte control word register, two-byte initial count register, and two-byte counter latch register.

- The actual counter is a two byte binary down counter, meaning it starts at some count N and decrements its count after each clock pulse it receives until N goes to zero, where the cycle then repeats. Getting a true count thus requires a subtraction, $real\_count = initial\_count - final\_count$. The counters are not directly user accessible!
- The control word register tells each counter what it is, and how it should behave. Control words must be sent to initialize the counters, and should be sent after each data collection run to ensure proper performance. See the 82C54 datasheet for details on control words.
- The initial count register contains the upper limit on the counter's counting. As above, the counter starts at N, where N is stored in the initial count register. For our purposes, N should be set to maximum, 0xFFFF, or $2^{16} - 1$. When two internal counters are cascaded, the initial count is then 0xFFFFFFFF, or $2^{32} - 1$.
- The counter latch register allows the user to access the current count of the counter without disturbing the normal operation of the counter. Once a counter latch command is sent to the control word register, the actual count of the counters is loaded to the counter latch register, where it can then be read at the user's convenience.

IMPORTANT: When new counts are written to the initial count register, they are not written to the actual counters *until a single clock pulse has been received* by the counter in question! For counter0 cascaded with counter1 on a given chip, this means that just after initial counts are written, counter0 must wait until after the first fringe pulse to begin counting, and counter1 must wait until it receives a pulse from out0 to begin counting. But out0 doesn't go low until counter0 reaches 2^{17}, so that number is essentially lost. This detail was overlooked in the original wavemeter design, which resulted in a systematic error that is now corrected in software. This is accomplished by adding 0x00020000, or 2^{17}, back into the final count.

This problem could be fixed in hardware by dedicating one of the few remaining Ethernut I/O pins as an initializing clock pulse pin. Four OR gates would need to be placed directly before clock0 and clock1 of both 82C54 chips. One input of each OR gate would connect to the initialization signal from the Ethernut. The initialization pulse would need to be added to counter.c after the two instances of counter_write_count() during cart turnaround.

The counter bus experiences some stray capacitance. When a pin is set *HIGH*, it rises immediately. But when it is set *LOW*, it has a rather long fall time. This was causing the Ethernut to read some erroneous counts. For this reason, $1k\Omega$ pull-down resistors were placed on each counter bus line. The Ethernut was also reprogrammed to clock the RD and WR pins 1ms after the outb() command so the data lines have time to settle. Since then no read errors have been observed.

4.4 Photodetector-Triggered Flip-Flop

Photodetectors are positioned at the ends of the track to sense when the cart has made a full translation. The mainboard provides power to and receives a signal from the photodetectors. The signal is normally *HIGH* to indicate the sense beam is unbroken. The signal goes *LOW* when the cart's flag breaks the beam. A comparator digitizes this signal, and the two digital turnaround signals are fed into a simple NOR flip-flop. This flip-flop controls the direction of the DC motor by actuating one of the two large motor relays.

The direction of the cart is, therefore, controlled *exclusively by hardware*. The Ethernut has no control over cart direction. This is to prevent damage to the cart and motor should the microcontroller fail.

However, the Ethernut must also receive information that the cart has reached the end of its track. It does this by sensing the output of the photodetector comparators and waiting for a low signal. This has the potential to fail, and will do so if the Ethernut is busy during the ~0.2s for which the flag remains in the photodetector, blocking the beam. Care must be taken to ensure that the Ethernut is free to sense cart turnaround. Excessive periodic operations, such as updating the LCD screen every second, are to be avoided. If the Ethernut fails to sense turnaround, it will continue counting fringes until it does manage to sense a turnaround. The resulting ratio of counts should still be ok to use *in theory*, but will likely be bad due to noisy fringes at turnaround.

A solution to this problem is to have the Ethernut sense the state of the flip-flop rather than the state of the photodetectors. The Ethernut could perform a periodic test, every 25ms or so, to see if the state has changed. The following code could work for that solution, if needed. It would go into counter.c, near the bottom, after the initialization of the counter thread:

```
int ffstate;
ffstate = bit_is_set(flip_flop_output);
for(;;) {
  NutSleep(25);
  if (ffstate != bit_is_set(flip_flop_output)) {
    ffstate = !ffstate;
    (Put the rest of the cart turnaround commands here)
  }
}
```

4.5 LCD connections

The mainboard doesn't actually do any processing of the LCD control lines. It just reorganizes them to make ribbon cables easier to attach. One important note is that PD4-PD7 of the Ethernut *must* correspond to D4-D7 of the LCD interface in order for data masking to work properly. In models previous to wavemeter v2.2, the LCD data bus connected to PD3-PD6 of the Ethernut, which is very difficult to work with in software.

Data masking allows the Ethernut to efficiently communicate with the LCD screen in 4-bit mode. For speed, we want to be able to issue the command outb(LCD_PORT, 0xYZ), where YZ is a hexadecimal number corresponding to an 8-bit pin configuration. We want to use this rather than a messy series of sbi() and cbi() commands. However, only the upper 4 bits, Y (D4-D7), are used to transfer data to the LCD. We want the configuration of the lower 4 pins, Z (D0-D3) to remain the same when we issue the outb() command. A data mask of 0xF0 accomplishes this task.

Data masking works by combining three bytes, two of which contain the desired data, and one which acts as a map of how to combine them. (old_byte & ~data_mask) | (new_byte & data_mask), where "~" indicates the bitwise inverse, results in a hybrid byte which contains the bits of the old_byte in the positions of zeros in the data_mask, and the bits of the new_byte in the positions of ones in the data_mask. So if new_byte = ABCDEFGH, old_byte = abcdefgh, and data_mask = 11110000, then the masked data will be ABCDefgh.

5 Ethernut

The Ethernut is a sophisticated web-enabled microcontroller that makes the wavemeter possible. The Ethernut performs all communication with peripheral devices, timing, and calculations. The best way to understand the code is to dive in. I've tried to add some concise comments and reorganize some sections of code to make it flow a little more logically, and there are some helpful command notes at the end of this document.

Ethernut code (written in C), can be found in the directory:
zoinks@atomoptics.uoregon.edu:~/zoinks/enut2app/wavemeter

5.1 Pin-Space

In order to communicate with all the different components of the wavemeter, the Ethernut uses almost every available I/O pin. Free pins may be found by cross-referencing the definitions in wavemeter.h with the Ethernut 2.1 Hardware Manual, pages 23 and 24. Currently only PD3, PE4, and PF3 are available for use. PD3 is currently defined as the LCD_BACKLIGHT pin in wavemeter.h, but as explained in Section 6, wavemeter v2.2 doesn't actually use this pin. The functionality of PE4 and PF3 have not been tested. See Page 22 of the Ethernut 2.1 Hardware Manual under the heading "Expansion Port" for more information.

5.2 IC11 - Serial Flash Memory

In order to easily communicate with the counters using the outb() and inb() commands, all eight pins of Port B of the Ethernut were assigned as the counter chip data bus. However, PB4 acts as the chip select for the Ethernut's serial flash memory, for which data are communicated on PB3. So whenever the Ethernut attempts to transfer data on PORTB that involves PB4 *HIGH*, PB3 drops to about 1V and fails to send/receive any data. To prevent this from happening, the signal from PB4 to the serial flash memory must be cut, which is fine because *the wavemeter does not use serial flash.* This can be achieved by completely removing the tiny IC11 from the Ethernut board, or by cutting the trace to pin 1 of IC11 which comes from a via very close to that pin. This isolates serial flash and allows the Ethernut to communicate with the counters properly.

5.3 MAC Address

The Ethernut MAC address is the only networking value that is stored in non-volatile memory. The other values are hardcoded. If for some reason the MAC address is wiped, the current Ethernut board's original MAC address is 00:06:98:21:3C:80, which is registered with the IT department so you kind of have to use it. However, you will be unable to communicate this to the Ethernut over telnet since it has no MAC address. You will have to temporarily hardcode the MAC address to the Ethernut. Look for instructions for this contingency in wavemeter.c.

5.4 Burning Code to the Ethernut

waldo.uoregon.edu is the computer that is physically near the wavemeter, and all attempts to install the C compiler onto Waldo have thus far failed. Even though we installed the compiler onto Waldo, the machine refuses to compile any of the wavemeter code. It just spits out a bunch of errors. I suspect that it's missing quite a few custom libraries, probably ones that are Ethernut specific. Someone can fix this if they want, but there are ways around the problem.

To burn code to the wavemeter:

- Disconnect the two ribbon cables running to the mainboard from the Ethernut. These interfere with burning for reasons unknown. You may leave the ethernet and power cables plugged in.

- Connect the Ethernut to the JTAG adapter, which connects to the serial cord, which connects to the serial-to-USB adapter, which connects to the USB port of Waldo.

- Waldo must have a driver for the Keyspan Serial-to-USB adapter. Go to terminal, then *cd ../../dev*, then *ls*. You should see something like *tty.KeySerial1* in the list of devices when the adapter is plugged in. If you don't, you must download and install drivers from the manufacturer website.
- Supply power to the Ethernut. It won't communicate with JTAG unless it has its own power. Powering up the Ethernut should cause the green light on the JTAG adapter to turn on.
- The JTAG adapter does not interfere with Ethernut operation until it is actually burning code to the microcontroller, so you may leave it connected during normal wavemeter use if desired.
- *ssh zoinks@atomoptics.uoregon.edu* and *cd zoinks/enut2app/wavemeter*
- Enter "*./compile.pl*" to run a simple script which compiles the code on atomoptics, then sends the result to Waldo.
- On Waldo, *cd zoinks/enut2app/wavemeter* and type "*make burn*". This should activate the burning process. If any errors appear, first check to make sure the JTAG cable is connected properly. If that doesn't solve the problem, recompile the code and try to burn again. If that doesn't work, try using magic.

6 LCD Display

The LCD is more of a rough indication of good operation than a necessity. Any real data collection is eventually going to be handled by the wavemeter website or ZOINKS anyway. The LCD is currently programmed to display a welcome screen at power-up, then update with pertinent information just after the Ethernut senses cart turnaround. The information includes the actual, corrected fringe counts of the reference and test laser beams, as well as the user-defined reference wavelength and the calculated test beam wavelength.

In order to calculate the test beam wavelength, the user must provide the wavemeter with the reference beam's wavelength. *This is done through telnet.* Access the LCD Debug menu and select "reference". Inputting a new number here stores that value in the Ethernut's non-volatile memory so the Ethernut will continue to use that value, even after reboot, until a new value is provided.

The LCD model used in the wavemeter is different from the type used in the WebTC boxes. The wavemeter's DMC-20481-NY-LY-ABE has 4 lines for display rather than 2 in the WebTC. This caused some confusion with addressing the characters, since the LCD module's internal control circuitry treats the 4-line by 20-character display as a 2-line by 40-character display. The details on this can be found in the Optrex LCD DMC User's Manual on the wavemeter project page.

The signal timing is sufficiently different between the two LCD models that an additional Ethernut function, lcd_busy() in lcd.c, was required to successfully communicate with the LCD module. This function reads the LCD module's "busy flag", which is D7 high when the module is performing an internal operation. While the flag is high, the Ethernut waits up to 40ms to send additional information.

The LCD screen starts up in 8-bit mode and must be initialized for 4-bit operation before use. This conserves the Ethernut's digital pins. The initialization procedure can be found in the Optrex DMC LCD Module User's Manual and is heavily commented in lcd.c.

The Ethernut currently has a I/O pin reserved for activating and deactivating the LCD backlight. In designs before wavemeter v2.2, this digital pin was connected directly to the backlight. However, the backlight requires several hundred milliamperes to operate and the Ethernut cannot supply that kind of power. The backlight is currently hacked to be always on by tying it to 5V through a 5W 8Ω resistor. The v2.2 Osmond files have been modified to reflect this hack.

To actually fix the problem, it will be necessary to modify the wavemeter mainboard Osmond files to accommodate either a digital power MOSFET or chip relay. The Ethernut would then run the gate or coil, respectively, which would control the current flow to the backlight. In all cases a 5W 8Ω resistor must be placed in series to limit the 5 volt supply.

7 PHP Server

In active mode, which is currently always on, the Ethernut sends final count data to the Rubidium server by PHP GET command. GET is a method for transferring information from a client to a server within a URL. The final counts stored on Rubidium are actually included in the URL that the Ethernut tries to access after cart turnaround, which looks like:

http://rb.uoregon.edu:1123/~rb/wavemeter/wavemeter.php?count0=X&count1=Y
Then wavemeter.php scrapes the values X and Y and does some simple operations with them. The actual file wavemeter.php exists on rb@rb.uoregon.edu:~/Sites/wavemeter, along with wavemeter.dat and several other useful items.

The GET command triggers the update of several items. First the counts are corrected due to issues explained in Section 4.3. The corrected counts are appended to wavemeter.dat. A Perl script, stats.pl, is run to compute some statistics based on the present contents of wavemeter.dat and reference.dat and store the result in stats.txt. A second Perl script generates histograms of the data in wavemeter.dat.

8 Wavemeter Website

The wavemeter website at http://rb.uoregon.edu:1123/~rb/wavemeter/ is where all the useful data ends up. Unless ZOINKS is somehow eventually used to control the wavemeter, users will be reading information from the website almost exclusively. The site is programmed in the file rb@rb.uoregon.edu:~/Sites/wavemeter/index.php.

The website is set to refresh itself every 10 seconds. The wavemeter takes about 30 seconds to translate the cart and generate a new data point, so the website is essentially a real time view of the measurement results. On every refresh, index.php reads in reference.dat, which contains the user-defined reference wavelength. A number entry field is provided to enter the wavelength of the reference beam. It also reads in and displays the contents of stats.txt, as well as the histograms and last three (most recent) lines of wavemeter.dat.

9 Other Notes

- In the Ethernut code, outp(data, port) is identical to outb(port, data)
- PORTX is the Ethernut register that sets the states of pins on some Port X. For example, if Port B pins are set as outputs, outb(PORTB, 0xFF) sets all pins on Port B to 5V.
- PINX is the Ethernut register that reads in the current states of pins on some Port X. For example, if Port B pins are set as inputs, inb(PINB) returns the current state of those pins as an 8-bit number.
- DDRX is the Ethernut register that controls whether a certain pin on a certain port is an input or output pin. For example, outb(DDRB, 0xFF) sets all pins on Port B as outputs, and outb(DDRB, 0x00) sets them all as inputs.
- NutDelay() is limited to 255ms and locks the CPU into a delay so it cannot perform operations on other threads. The delay time is precise.
- NutSleep() has essentially unlimited time span and allows the CPU to run other threads during the delay. NutSleep() is less precise than NutDelay().
- If you want to prompt the user for string input while using telnet, use the FetchLine() command, which is defined near the top of telnet.c. Traditional C commands such as scanf() won't work for some reason.
- The Ethernut's NutOS does not allow direct printing of float or double numbers using the %f formatting tag. These must be converted to strings manually using the dtostrf(double value, char width, char precision, char * string) command. The string may then be printed as a %s.
- The "testfreq" command in telnet is a pretty sketchy. It sort of works, but don't trust its accuracy. It uses NutSleep(1000) to measure the difference in counts over 1s. NutSleep isn't very accurate, and testfreq doesn't know to throw out data collected over a cart turnaround, for instance.