# WASP and APHID Microcontroller Command Interface
## BIMA Memorandum #83

A.I. Harris
Department of Astronomy, University of Maryland
18 November 2003, code version wasp223

Each WASP and APHID spectrometer contains an Intel 80C251SB microcontroller that manages the spectrometer's real-time and other hardware tasks. In single-dish observing the microcontroller would, for instance, accumulate data from the spectrometer hardware, send signals to the telescope's chopping secondary and beamswitch ("nodding") inputs, and set the spectrometer's instrument state. It communicates with a host computer or terminal using simple commands and a serial interface. This memorandum describes the microcontroller commands.

The basic logic for spectrometer control is that the host should command the microcontroller for a single completely specified action, leave the microcontroller to its task unattended, and recognize that the task is finished when the microcontroller sends its attention signal. In this way, the host is always certain of the task's setup. This simplifies the programming interface and allows the host to communicate with many spectrometers without conflicts. The price for this is small: a few extra characters are needed to initiate tasks in a specific state. The command set is deliberately small. Most of the hardware-specific mode settings are programmed in the microcontroller rather than set as options by the host computer.

A typical command sequence would be for the host computer to send a command for a chopped and nodded astronomical integration with a string specifying the chopping, nodding, and blanking variables: `n 80 0 100 10 4 400`; wait for the microcontroller to finish and send its attention signal: `!`; and then read back the data: `s 1024`. If the microcontroller detects an error state it returns a single character in place of the `!`. Table 1 is a summary of the possible commands, and Table 6 summarizes the error states and messages.

All commands to the microcontroller are case-sensitive and end with a `<return>`. In "terminal" mode the data return as alpha-numeric ASCII characters; in "computer" mode the data return in signed (two's complement) 32-bit integer words. The word is composed of 4 bytes with the most significant byte sent first. The basic time unit for the commands is the frame rate of spectrometer readouts, a time base of 11.520 ms. The serial port setup is RS-232, 19.2 kbaud, 8 bits, one stop bit, no parity. In principle the transfer obeys Xon/Xoff, but this has not been thoroughly tested. Enabling Xon/Xoff at the host end is inadvisable since the control characters (0x11, 0x13) may easily occur during a binary transfer.

## 1 Command Listing

### 1.1 *Help,* `h`

Send the table containing the command menu in terminal mode with `h`. There is no action in computer mode.

### 1.2 *Computer or terminal,* `d`

Set computer and terminal modes with `d mode`, where `mode = 0` is computer and `mode = 1` is terminal mode. The default at boot is the terminal mode. In terminal mode, the output is designed to be read by a human: it is more verbose and the numbers are in decimal format.

| Action | Command | Function |
|---|---|---|
| Help | h | help: print menu |
| Term/Comp | d # | select computer (0) terminal (else) |
| Zero | z # # | internal offsets, n_fra cycles, mode 0-3 |
| Level | l # # | set level in lag to \|val\|; set atten # dB |
| TotPwr | t # | sum data for n_fra frame pair |
| Chop | c # # # # | n_fra, nodside, chops, c_wait |
| Nod | n # # # # # # | n_fra, nodside, chops, c_wait, nods, n_wait |
| Halt int. | (any key) | any key stops integration |
| Send data | s # | send (nbytes of) accumulated data |
| Stats | m | mean, variance |
| O'scope | o # # # | time seq: adc, n_sum, n_sam |
| Nod tel | f | toggle nod state |
| Ctrl byte | b # | write chop, nod signals if 1; slave if 0 |
| Init ADCs | i | send ADC setup string |
| Query | q # | query # chans of instrument info |
| Switch | x # # | write to switch #, state=1,0 |
| Write | w # | write # to DAC |
| APHID | a # | start APHID readout mode with # reads/cycle |
| Eval | e # | evaluation modes (0 for normal operation) |
| Version | v | version of code |

Table 1: Command list for the WASP and APHID spectrometers.

In computer mode, the communication is cut to the minimum: commands are not echoed, and the data are transferred in binary format.

### *1.3 Zero,* `z`

Find internal electronic offsets over an integration time of length `n_fra` readouts and in mode 0, 1, 2, or 3 with the command `z n_fra mode`. Setting `mode` to 3 switches the microwave power off with a switch inside the amplifier module, measures the zero level, and then returns the attenuation to its original setting. In `mode` 0, 1, or 2, zero is measured by shutting off the usual microwave phase switch clock. In mode 0, the mixer is set to phase 0; mode 1 puts the mixer in phase 1; and mode 2 is the average of both phases. Phase modes other than 0, 1, 2, and 3 return zeros. (Note: if n_fra is not even in phase mode 2 (`n_fra, 2`), the number of readout cycles will be rounded to the next lowest integer.) Dividing the sum in the output buffer by `n_fra` produces the average offset value. The readout frame time is 11.520 ms, so the integration time $t_{int,z}$ is:

$$t_{int,z} = \mathtt{n\_fra} \times 11.520 \text{ ms} . \tag{1}$$

Settling and synchronization times add a few times 11.520 ms to $t_{int,z}$ for the elapsed time. The result will be 128 signed 4-byte words in ADC readout order (not time lag order) in the data buffer.

### 1.4 *Level,* `l`

Use the attenuator in the amplifier module to auto-set the power level in adc number `adc_no` for |`val`| counts per readout with `l adc_no val`. For instance, `l 3 18000` will set the absolute value of the level in adc number 3 to 18000 counts per readout, if this is within the attenuation range. Note that this routine works on the raw adc number rather than the (reordered) lag numbers, so using the `m` or `t` command is a good way to find the zero lag. Attenuation values between 1 and 15 dB are preferred for band flatness. The routine returns values for attenuation, final value, and requested value after the lag readout data in words 128, 129, and 130 of the data buffer. The routine returns an error state if the final level is not within a factor of 0.7 (1.55 dB) of the requested level.

Setting the adc number to values between 600 and 699 sets the attenuator value directly with an offset of 600. For example, `l 600` sets the attenuator to 0 dB and `l 609` sets it to 9 dB. The attenuation range is 0–31 dB; requesting higher values activates a switch in the module that provides more than 60 dB attenuation.

Setting the adc number to values between 700 and 799 is reserved for level setting using an external power detector read by one of the system ADC channels. This has not been implemented yet.

### 1.5 *Total Power,* `t`

Accumulate total power data from the spectrometer for `n_fra` readout frames with `t n_fra`. The readout frame time is 11.520 ms, so the integration time $t_{int,t}$ is:

$$t_{int,t} = \texttt{n\_fra} \times 11.520 \text{ ms} . \tag{2}$$

The output will be 128 signed 4-byte words in ADC readout order (not time lag order). Because the microcontroller synchronizes itself to the readout frame clock, there will be an uncertainty between 0 and 11.520 ms in the actual start time after the microcontroller receives the command.

See the control byte section to set a mode for writing data from previous integration during current integration. This can give high efficiency on closely spaced short total power integrations.

### 1.6 *Chop,* `c`

Accumulate chopped power data from the spectrometer for `n_fra` readout frames per chop side, into even or odd nod side `nodside`, for a total of `chops` chop cycles, and a chopper transition blanking time of `c_wait` frames with `c n_fra nodside chops c_wait`. The chopping frequency is

$$f_{chop} = \frac{1}{2(\texttt{n\_fra} + \texttt{c\_wait} + 1)11.520 \text{ ms}} , \tag{3}$$

and the integration time on the sky is

$$t_{int,c} = 2 \times \texttt{chops} \times \texttt{n\_fra} \times 11.520 \text{ ms} . \tag{4}$$

The nodside variable determines whether the data are stored in the first 128 words of the data buffer (even, 0) or the second 128 words (1, odd). Synchronization at the beginning of each chop side takes 11.520 ms, accounting for the additional 1 in the equation for $f_{chop}$ (remember to absorb this extra readout in the chopper blanking time). Because the microcontroller synchronizes itself to the readout frame clock, there will be an uncertainty between 0 and 11.520 ms in the actual start time after the microcontroller receives the command.

### 1.7 *Chop and nod,* `n`

Integrate with chopping and nodding. This command builds on the chopping command with the addition of the number of nod cycles `nod` and telescope move blanking time `n_wait`: `n n_fra nodside chops c_wait nod n_wait`. The integration time on the sky is

$$t_{int,n} = 2 \times \texttt{nods} \times t_{int,c} = 4 \times \texttt{nods} \times \texttt{chops} \times \texttt{n\_fra} \times 11.520 \text{ ms} . \tag{5}$$

The blanking time for telescope moves is

$$t_{wait}(tel) = \texttt{n\_wait} \times 11.520 \text{ ms} . \tag{6}$$

The elapsed time, including blanking and synchronization time overheads is

$$t = (4 \times \texttt{nods} \times \texttt{chops} \times (\texttt{n\_fra} + \texttt{c\_wait} + 1) + (\texttt{nods} + 1) \times \texttt{n\_wait} + 1) \times 11.520 \text{ ms} . \tag{7}$$

The telescope always nods at the beginning of an integration to insure that the telescope has a well-defined pointing for chopping and nodding. Without this initial nod, the telescope may point between the two beams; this will always the case for a new source. The `nodside` variable sets the beginning nod state, even or odd. The data are stored as 256 words, with the first 128 containing data from the even nod side, and the second 128 from the odd nod side. Synchronization at the beginning of each chop side requires 11.520 ms, accounting for the first 1 in the equation for elapsed time. Because the microcontroller synchronizes itself to the readout frame clock, there will be an uncertainty between 0 and 11.520 ms in the actual start time after the microcontroller receives the command, the second 1 in the equation.

### 1.8 *Halt integration,* `(any key)`

Sending any character to the microcontroller will halt the integration and return it to its listening mode, awaiting the next command. The microcontroller retains the characters that halted the integration and will act on them as soon as `<return>` is in the string.

### 1.9 *Send data,* `s`

Send `nbytes` bytes or words from the data buffer with `s nbytes`. The data buffer words are signed (two's complement) 4-byte (32-bit) integer words. In computer mode `s` transmits`nbytes` bytes starting with the first byte in the buffer, then the second, and so on. In terminal mode, `s` transmits `nbytes` words (ADC values) in decimal form. At 19.2 kbaud, the transfer time for 256 words is about 0.5 sec in computer mode. The transfer time in terminal mode depends on the values and is unpredictable.

### 1.10 *Statistics,* `m`

Compute the mean and variance for 32 consecutive readouts of the input data for all channels. The mean is in the first 128 words of the data buffer, the variance is in the second 128. The data are in order of ADC readout, not time lag.

### 1.11 *Oscilloscope,* `o`

Record a time sequence data for one lag channel. `o adc n_sum n_sam` takes `n_sam` data points from ADC `adc`, summing `n_sum` frames at each datum. The lag selected is in order of ADC readout, not time delay. As an exception in the oscilloscope mode, data always return without prompting in ASCII mode in two columns: sample number and data. Table 2 gives the lag-to-ADC correspondence: That is, the first value from the microcontroller is from lag 1, the second

| ADC | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Lag (A) | 1 | 3 | 5 | 7 | 9 | 11 | 13 | 15 | 14 | 12 | 10 | 8 | 6 | 4 | 2 | 0 |
| Lag (B) | 14 | 12 | 10 | 8 | 6 | 4 | 2 | 0 | 1 | 3 | 5 | 7 | 9 | 11 | 13 | 15 |

Table 2: Readout order corresponding to time lag for cables connected in the A and B configurations.

is from lag 3, and so on. In WASP2 the zero time lag falls near lag 2 (ADC 14) or 1 (ADC 0).

To get data into a file `tser.dat` on the host computer, use a command like

```
serial 0 "o 14 9 2520" timeout=260 out=tser.dat
```

Table 3 gives a list of useful setups for Allan variance measurements. Series with `n_sam` in multiples of 360 give particularly useful series of data for Allan variance analysis. The integration time for each sample, in seconds, is

$$t_{int} = \texttt{n\_sum} \times 11.520 \text{ ms} , \tag{8}$$

the elapsed time for the entire series, in seconds, is

$$t_{elapsed} = t_{int} \times \texttt{n\_sam} , \tag{9}$$

and the timeout time for the receiving computer is `timeout` seconds, which should be slightly longer than $t_{elapsed}$.

| $t_{int}$ (sec) | `n_sum` | `n_sam` | elapsed (min) | timeout |
|---|---|---|---|---|
| 0.104 | 9 | 2520 | 4.4 | 265 |
| 0.254 | 22 | 1080 | 4.6 | 275 |
| 0.507 | 44 | 1080 | 9.1 | 550 |
| 1.003 | 87 | 1080 | 18.1 | 1085 |
| 5.004 | 434 | 720 | 60.0 | 3610 |

Table 3: Useful setups for Allan variance measurements; good number of samples are 1080, 2520, 5040.

### *1.12 Nod telescope,* `f`

Nod the telescope back and forth between beams for test and setup purposes with `f`. The nod pulse is 4 seconds long.

### *1.13 Control byte,* `b`

Switch the signal direction to write (`b 1`, master) or read (`b 0`, slave) the synchronization signals Chop, Nod A, Nod B, and Blanking, to or from the external port. The default direction is set by a compile-time variable in the microcontroller code, so this is a way to change from the default.

Using `b 3` (master) and `b 2` (slave) is a high-efficiency mode for total power integrations (not chop or nod) that provides a binary write of the previous integration's data during the integration. Since this mode writes data from the previous integration, the first readout may or may not contain useful data. Approximately 23 bytes are written each 11.52 ms cycle, so the minimum number of readout cycles should be 23 (265 ms) for full efficiency for APHID, or 184 cycles (2.2 s) for WASP. All data will be written regardless of the integration time. Data may also be read with the `s` command, with data alternating between the lower and upper halves of the data buffer.

Caution: a 1 k$\Omega$ current limiting resistor should buffer the signals; excessive current may flow if a low-impedance external signal is connected to the buffer when it is the master.

### *1.14 Initialize ADCs,* `i`

Send the setup string to the spectrometer's analog to digital converters with `i`. The micro-controller does this (with an approximately 2 second delay to allow power supplies to settle if needed) on power-up and hardware reset, but this command is a soft way to reset the ADCs if the microcomputer and spectrometer have different power supplies or if the ADCs seem upset by static discharge or some other problem.

### *1.15 Query,* `q`

Query instrument state channels 0–`q`; e.g. `q 2` will get information for the first two channels, 0 and 1. Get the data by sending the `s` command for the corresponding number of channels. The value returned in the first channel is the input attenuation in dB. Subsequent values are sensor voltage in mV from the internal ADC. For the LM50C thermal sensors in the system, the temperature is

$$\mathrm{T}(^\circ\mathrm{C}) = \frac{V_{sensor} - 500 \text{ mV}}{10} , \tag{10}$$

so 773 mV corresponds to 27.3°C. Table 4 is a list of the sensor channel numbers and descriptions.

| Sensor | WASP | APHID |
|---|---|---|
| 0 | Input attenuation [dB] | |
| 1 | Last corr card | Amplifier plate temp. |
| 2 | First corr card | Correlator board temp. |
| 3 | Avg. corr nos 1, 6 | |
| 4 | Avg. inner 4 corr cards | |
| 5 | Amplifier module | |
| 6 | Interface card | Ambient (ADC temp.) |
| 7 | Unbuffered ext. inp. | |
| 8 | Buffered ($\times -1$) ext. inp. | Buffered ($\times -1$) ext. inp. |

Table 4: Sensor channel numbers and descriptions.

### *1.16 Switch,* `x`

Send commands to drive switches for external devices with the `x` command. The first variable denotes the switch number (0–3), the second its state (0 or 1). A state other than 0 or 1 will set all switches off. The switch driver is a Texas Instruments TPIC6595 part which can switch ground-returned voltages to 45 V, 1.5 A pulsed current, and 250 mA continuous current. The drivers have internal snubbing for inductive loads such as relay coils. Table 5 is a list of the external switch numbers and functions.

| Switch | WASP | APHID |
|---|---|---|
| 0 | | |
| 1 | | |
| 2 | | |
| 3 | | |

Table 5: External switch numbers and functions.

### *1.17 Write,* `w`

Write a number to the DAC. The output voltage will be this number in millivolts, with a range of 0 to 4096 mV.

### *1.18 APHID,* `a`

Start continuous readouts of the first 16 lags in the APHID spectrometer mode with `a m`. The argument `m` is the sky integration time in units of 11.520 ms readout times. Data from the previous integration write during the current integration. Setting $\mathtt{m} \geq 3$ allows time for the full write during integration. All data are written back even if the integration time is shorter than the write time. The voltage from a temperature sensor, measured at the beginning of the sky integration, returns in word 17.

No additional readout times are necessary for the temperature measurement and data transfer, so the total time per integration cycle is:

$$t_{cycle} = 46.1\text{ms} + (\mathtt{m} - 3) \times 11.520 \text{ ms} . \tag{11}$$

In the internally-triggered (master) mode the next integration begins immediately after the previous cycle finishes. If the microcontroller is in slave mode for synchronization (`b 0`, see `Ctrl Byte`), the readouts start on the first rising edge at the chop signal port after the command has been received. Exit from the continuous readout mode by sending any character to the microcontroller; the character is retained but not acted upon until the string terminates with a `<return>` character.

### *1.19 Evaluation modes,* `e`

A number of evaluation modes are available through the `e #` command, where `#` is a mode number. The mode is held until it is reset by the `e` command or a hardware reset. Specifying mode `0` or an undefined mode number results in normal, default operation. Current modes are:

#### 1.19.1 Evaluation mode 0, default

Normal operation.

### 1.19.2 Evaluation mode 1, raw ADC data

Return the raw ADC data, summed but not software differenced. With no microwave signal the command `t 1` and then `s 128` should return the ADC input bias values near $2^{15} = 32{,}786$. The format is values for the two readout phases in adjacent columns.

### 1.19.3 Evaluation mode 2, generate data pattern

Generate a test data array to check serial interface and endianism; return data with `s #`. The data array starts with a 4-byte word containing the numerical value 1 as an endianism check, then increments succeeding bytes in the array as 0x01, 0x02, 0x03, ..., 0xff, 0x00, etc. The 80C251 microcontrollers are big-endian: the most significant byte is stored at the base address, with remaining bytes at higher addresses, to the least significant byte at the highest address. This mode is also useful for testing error rates on the serial link.

### 1.19.4 Evaluation mode 3, ADC noise evaluation

Shifts reads by 3 bits lower (factor of 8) for a closer examination of the ADC noise. Since the width of the accumulated word remains at 16 bits, the maximum signal before overflow is also a factor of 8 lower.

### 1.19.5 Evaluation mode 4, DAC test

Write DAC value from input string for testing DAC. In the default mode, the DAC value is written from the specified ADC, but this has not been fully tested.

### 1.19.6 Evaluation mode 5, modulate chop pattern

Chop signal sense flips depending on nod side: normal for nod A, complement for nod B. This may be useful for modulating a test source in simulating long integrations. The mode operates only on the chop signal and not on the internal buffer selection. The micro must be in the boss mode `b 1` for this mode to have any effect.

## *1.20 Version,* `v`

Send a message with information about the version number and build date. This information is updated manually in the `wasp2.h` header file in the microcontroller programming environment.

# 2 Synchronization to external timing signals

In slave mode (`b 0`) the microcontroller starts integrations in response to external timing signals. These signals are:

- Blanking, a data blanking line which is low for valid data, high for blanking times.
- Chop, the secondary mirror chop phase.
- Nod A, which pulses high when the telescope moves to the A beam (100 ms minimum, telescope slew time maximum).
- Nod B, which pulses high when the telescope moves to the B beam (100 ms minimum, telescope slew time maximum).

These signals and a representation of the actual secondary mirror position are shown schematically in Figure 1. The logic for total power, chopped, and chopped and nodded observations follow below.

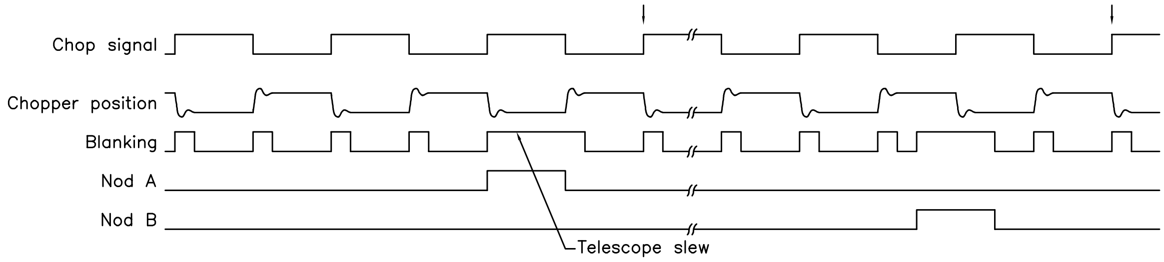


Figure 1: Schematic view of external timing signals. Integrations after a nod begin at the times marked by the small arrows above the chop signal.

### *2.1 Total power*

Integration starts on the falling edge of the blanking signal. A high blanking signal at the end of an integration indicates that the integration time is too long and generates a blanking timing error; the actual integration time must be slightly shorter than the interval with the blanking signal low to assure proper synchronization.

### *2.2 Chopped*

Integration starts on the first rising chop signal edge following the blanking signal's falling edge. Again, the actual integration time must be slightly shorter than the time in each chop side for proper synchronization. The blanking line should rise at the beginning of each chopper signal and remain high through the switching and settling time.

Trapped error conditions are blanking line high at end of integration (blanking timing error) or chop phase different at the beginning and end of an integration (chop phase error). An untrapped error is if the timing is so badly off that data accumulate through the following phase and two blanking phases. This error will be obvious from the overall timing: an integration will take somewhat longer than twice the requested duration.

### *2.3 Chop and nod*

Chopping and nodding adds two additional nod signals, one for each nod position (A and B), to the chopping mode. To initiate integration in either the nod A or nod B position, pulse the corresponding signal high. The pulse duration must be at least 100 ms and at most the telescope slew time for the nod. Data acquisition begins within a chop time after the blanking signal falls, so the blanking signal must remain high during the slew and settling times. The actual integration time must be slightly shorter than the interval with low blanking signal. The system assumes an alternating nod pattern with intermediate slews left out for efficiency (e.g. for three nods: A, slew, B, B, slew, A, A, slew, B). Trapped errors (nod timing error) are generated if either nod signal is high at the end of an integration or if both are high at the same time. Other errors can be generated by chop mistiming (sec. 2.2). Repeatedly nodding into the same beam is an untrapped error that can be detected by comparing the two nod buffers. Values in the two buffers should be approximately equal when the nodding pattern is correct.

## 3 Error messages

Table 6 summarizes the error messages that the microcontroller can return.

| Computer mode | Terminal mode | Explanation |
|---|---|---|
| ! | WASP2> | No errors detected |
| S | WASP2 serial line interrupt> | Serial line activity during integration |
| O | WASP2 overflow error> | ADC overflow |
| B | WASP2 blanking timing error> | Blanking line high at end of integration |
| C | WASP2 chop timing error> | Chopper in different phase at end of integration |
| N | WASP2 nod timing error> | Nod signals both high or nod signal high at end of integration |
| D | WASP2 data stream not detected> | Data stream from ADCs not detected at startup |
| L | WASP2 atten loop did not converge> | Auto-tune attenuator loop did not converge within limit |
| ! | WASP2 (ERROR 0x??)> | Hex value of error not defined above |

Table 6: Error list for the WASP and APHID spectrometers.

## 4 Miscellany

### *4.1 Data stream detection*

The microcontroller board accepts data from both a differential line receiver and a single-ended input. Data are permitted on both inputs simultaneously, but the microcontroller will read only one. On boot, the microcontroller first tries to find a data stream at the differential receiver. If that fails, it tries the single-ended input. If that fails, it leaves the input select at the single-ended input and returns with an error message that no data stream has been found. Port information is available in the data buffer: the first value is the datum used to select the port, and the second value is the port number: 1 for differential, 2 for single-ended.

### *4.2 Downloading code to the Phytec microModul251*

The microcontroller's flash memory loads over the serial port with the following steps:

- Connect a Windows PC to the serial port.
- Simultaneously press both buttons (reset and boot) at the edge of the microcontroller board.
- Release the inner (reset) button while continuing to hold the outer (flash) button.
- Release the outer button.

- From the directory with the microcontroller and flash tools code, run the script `fl.bat`. This invokes the Phytec flash tools with the string `flasht br (19200) 1`. If the Windows PC's serial port is not COM1, replace the last 1 in the string with the appropriate port number.
- A DOS screen should appear with messages from the microcontroller. Choose option 1 (64 kB memory), then option 6 (erase and load), then y to confirm the erase.
- When the flash has been erased, hit the function key F2 to enter the file name. Type `wasp2.hex` at the prompt for the file name. The code will spill down the screen as it loads. Distracting the Windows PC during the download may cause the download to fail.
- When the download is complete, reboot the microcontroller with the reset button (inner) or cycle the power. The DOS screen will function as a terminal emulator.
- Hit the function key F1 to exit from the flash tools. Close the window and disconnect the Windows PC.
- Reconnect to the usual host PC.