HARDWARE HACKER

More on laser printing and bar codes, how stream gauges work, 256K RAM upgrades, and understanding ASCII control codes.

By Don Lancaster

Laser printers are such a major breakthrough in personal publishing power that there is absolutely no way they can be ignored. To quote the obvious: "The power of the press lies in owning one." Yes, their prices are still a tad on the steep side. But prices are dropping dramatically, and there are unlimited opportunities out there. These opportunities include both custom local printing, as well as showing others what can be done with these landmark products. I've got a free demo pack showing that I'd be pleased to send you. Just call or write.

Here's a little more info on bar codes, over and above what we looked at two months ago. There's apparently a magazine called *Bar Code News.* They have separately published a book called *"An Introduction to Bar Code Technology"* that sells for $16.95. Authors are Craig Harmon and Russ Adams.

Another new book I just received is called *The Commodore Ham's Companion*, self-published by Jim Grubbs (L9E1) at *Q-Sky Publishing.* Included are lists of suppliers, organizations and magazines of interest to both ham radio operators and Commodore owners.

On to this month's goodies . . . .

How does a Stream Gauge Work?

Ben is the honcho at our local United States Geological Survey (USGS) office. Every once in a while, he calls with an offer I just can't refuse. Such as "Let's go to Lower Eagle Creek for the day."

You see, it is Ben's job to maintain the stream gauges that accurately measure the water flow in Southwestern canyons, washes, stream, and rivers. Knowing how much water you get when is crucial in any arid land. Stream flow measurement in any area is very important for such things as flood warnings, dam and lake supervision, irrigation allotments and water resource management in general.

So how does a stream gauge work?

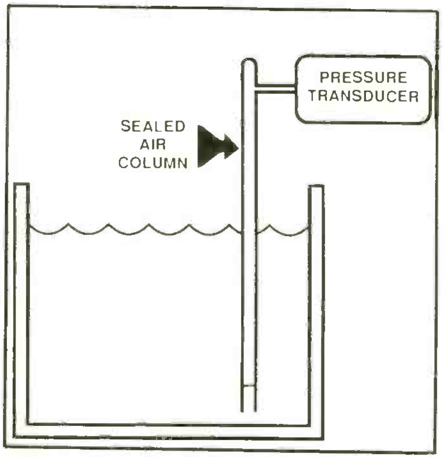

Fig. 1. This example of a liquid-level sensor has no moving parts.

On a small creek, all you need is a concrete structure called a "V-shaped weir." This is nothing but a flat channel with a Vee-bottom. Given the viscosity and temperature of a fluid, the flow rate will be proportional to the depth of the liquid in the weir. All you have to do is measure water depth and then consult a flow rate table. You do have to keep the weir clean and mud-free. The results aren't too accurate when dealing with muddy water or flood runoff.

Things get much messier much faster when dealing with a river. Besides temperature extremes, you have to contend with mud, channel movements, vandalism, access, long term reliability and many other hassles.

The flow-measuring idea remains the same. You measure the depth of the stream. You also do what is known as a "stream profile" once each month or so. This means you wade out into the stream or use the cable car to drop a flow meter into the stream along a bank to bank grid. The flowmeter is nothing but a small propeller that closes a contact for either a beeper for a counter. Flow is measured as beeps or pulses per minute. Calibration depends on propeller size and shape.

Modern stream gauges use a novel water-measurement scheme we will look at shortly. It can be located in an unobtrusive location well above and fairly far away from the water.

Today's stream gauges are usually solar powered and directly "talk" to a satellite by way of a microwave link. This link gets briefly activated every time the satellite whips on by. This eliminates the need for a chart recorder. More importantly, you find out about a flood when it is about to happen, rather than a month later. Malfunctions are also instantly spotted, rather than waiting weeks to find out you forgot to wind the chart drive mechanism or replace the paper.

Back to our original question. How do you remotely and reliably sense the height of water in a stream?

Figure 1 shows us one approach. Its the old "put your finger over the straw" method. As water height changes, pressure in the pipe changes. Measure the pressure, do some calculations, and you have the stream height.

There are some obvious problems with this method. The air pressure in the pipe may change with temperature, or may simply diffuse through tiny leaks. Worse yet, if the pipe is much longer than the depth of the fluid, the percentage change in pressure you would get with a change in stream height would end up so low that you could not reliably measure it. Instead, Fig. 2 shows some refinements on the "finger over straw" stream height measurement.

So what does all this have to do with hardware hacking? Just this. These dudes go for $22,000 each!

You wouldn't believe the Rube Goldberg method currently used to actually measure the pressure. A big tank of mercury is run up and down a steeply slanted mechanical rack. A motor repeatedly moves the mercury up and down the rack to exactly balance the mercury pressure against the nitrogen pressure in the pipe. Since mercury is much denser than water, a few inches of vertical mercury motion corresponds to several feet of change in water height.

Complex electrical connections are made and broken every now and then to balance the mercury tank against the pipe pressure. The *position* of the mercury tank on the rack is then measured and either sent to a chart recorder or routed directly to a microwave satellite transceiver.

At one time, measuring small changes in small pressures was tricky and expensive. Particularly if you were to eliminate any "stiction" or "hysteresis" in your readings. In theory, such a klutzy servo system can be extremely accurate, as it tends to cancel out its own nonlinearities.

In this day and age, any hacker can go out and buy a 0-to-7.5-psi temperature-compensated pressure transducer for well under $30 in singles from *Motorola, Microswitch* or any of several others. In quantity, they cost less than $5.

So, what is the simplest and cheapest way you can come up with to measure pressure over a 0-to-7.5-psi range? How compact can you make it? How cheap? How low in power consumption? This is an exercise for your hacker skills. Put on your thinking cap and let us know what you come up with.

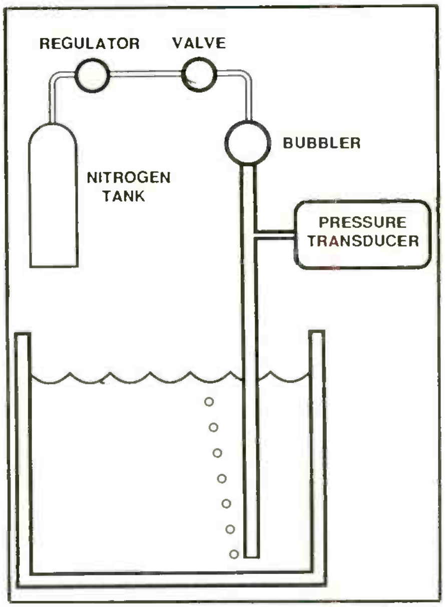

Fig. 2. This improved liquid-level sensor permits remote level sensing.

Upgrading from 64K to 256K RAMs

Your challenge depends entirely on the system you are trying to update. If the system was set up and designed for 256K RAMS in the first place, all you need do is change a jumper or two and directly swap the chips. For instance, the *Macintosh* upgrade is simply and easily done this way. The biggest hassle here is unsoldering the old chips. On other systems, an upgrade is probably far beyond the abilities of most beginning hardware hackers and definitely should *not* be attempted.

There are at least five major problems involved in expanding memory with 256K RAMs. The first and most obvious question is: "Where are the extra address lines going to come from"? If the rest of the system does not have the ability to address this much memory (most 8-bit systems do not), then where are you going to connect your new address lines?

One route to upgrading an 8-bit system is to use bank switching. With bank switching, you select one of four possible 64K banks of memory at any one time. But who is going to flip this bank switch? You or the CPU? How? How will you prevent the program you are reading from vanishing when a switch is flipped?

The second hassle with dynamic RAMs involves a concept called "pin multiplexing." There are not nearly enough pins on a 16-pin integrated circuit package to handle all of the address, data, control and power lines at once. So, many pins serve double duty and are used for two different purposes.

For instance, a typical address pin is time-shared between two address lines called the row address and the column address. A very elaborate, precise and time critical switching sequence is required. This address pin multiplexing can be handled by individual logic gates, by a special RAM controller chip, or sometimes by the microprocessor itself.

Adding an extra multiplexer for another pin can be a real hassle. Unless there is already enough address pin multiplexing provided for in the original circuit design.

Also, the "dynamic" in dynamic RAM means that the data will go bye-bye if you do not continuously refresh it. Many systems are not set up for refresh beyond 64K. It may take lots of extra parts and effort to provide for it.

Furthermore, you have to be sure that either the operating system or the individual software you want to run knows how to check and find out how much of this new memory is available. Otherwise, your new memory might just sit there unused, just out of reach.

Finally, there's the warranty problem. Many dealers refuse to service a system that has been obviously modified. Many customers will refuse to do such an upgrade for just this reason.

The bottom line: If you are an advanced hacker and are thoroughly checked out with an oscilloscope, you could possibly do a 64K to 256K expansion on your own. But do not expect to be able to do it simply, quickly or with nothing but a few chip swaps.

Leave this one alone unless you want to get into it in a really big way. Or wait until someone else offers a specific kit or detailed instructions.

The difference between a formfeed, ASCII $0C, CHR$(12), [L], FF, and <control-L>

This is really a software question, but it comes up over the help line so often that we really should look at it. It turns out that these are all *exactly* the same. That is, all are different ways of showing ASCII control commands. Control commands are the bottom 32 codes in the standard ASCII character code. These control commands are used for such nonprinting things as carriage returns, formfeeds, bells, escapes, linefeeds, and so on.

Unfortunately, there are many different ways of showing ASCII control commands. There are also many different ways of accessing them from a program. Figure 3 shows five of the more popular ways of naming ASCII control commands. It also shows how to access them from various programs.

The ASCII column tells you the mnemonic for each control command. For instance, CR is a carriage return, FF is a formfeed, BEL is a bell, and BS is a backspace. This mnemonic is a reference for people only, and is not recognizable by a personal computer.

The Hex Code column shows the same ASCII code in hexadecimal, as it would be accessed by a machine language program or subroutine. Here CR is a $0D, FF is a $0C, BEL is a $07, and BS is a $08. You are likely to see this hex notation when reading files associated with machine-language programs.

The Dec Code column shows the very same ASCII codes in decimal, as would be needed by Applesloth or another higher-level language. Here CR is a 13, FF is a 12, BEL is a 7, and BS is an 8. Many printer manuals show their control codes in decimal.

As decimal use examples, you might do a PRINT CHR$(13); CHR$(4) during a disk access to output a carriage return followed by a "<control-D>" DOS access command. Or, you might do a POKE 28756, 13 to force a carriage return into a text file. The 28756 is the address of the particular location in which you want to place the carriage return. This, of course, changes with your application. The 13 here is the actual carriage return.

As an imbedded printer code example, a PRINT CHR$(27);M will switch a Diablo 630 daisywheel printer into automatic microjustification.

The Apple Keys column shows how you would enter a certain control code from the Apple keyboard. As an example, a [L] means to press the control key, hold the control key down, press shift and L, release shift and L, and then release the control key. This gives you a formfeed entered into your text file or whatever.

Some word processors will want you to use a verbatim entry mode. For instance, to put a formfeed into an Applewriter file, press [V] [L] [V]. The first [V] says to begin verbatim entry. This means that what follows is to be put directly into the text file, rather than immediately acted upon. The [L] is the formfeed that gets placed into your file. The final [V] cancels verbatim entry so that any new control characters can actually be used for control purposes, rather than going into a text file for later use.

Finally, the Original Use column shows what the intended purpose of the control command was when ASCII was standardized. Most of the original uses still apply today. It would be extremely stupid to redefine a carriage return as anything else. On the other hand, the oddball and obscure commands, such as the group separator or range separator, are easily diverted to special commands that better suit certain personal computers or software routines.

ASCII	HEX	DEC	🍎KEY	ORIGINAL USE
NUL	$00	0	[@]	Do nothing or null
SOH	$01	1	[A]	Start of heading
STX	$02	2	[B]	Start of text
ETX	$03	3	[C]	End of text
EOT	$04	4	[D]	End of transmission
ENQ	$05	5	[E]	Enquiry
ACK	$06	6	[F]	Acknowledge
BEL	$07	7	[G]	Bell or alarm
BS	$08	8	[H]	Backspace
HT	$09	9	[I]	Horizontal tab
LF	$0A	10	[J]	Line feed
VT	$0B	11	[K]	Vertical tab
FF	$0C	12	[L]	Formfeed
CR	$0D	13	[M]	Carriage return
SO	$0E	14	[N]	Shift out
SI	$0F	15	[O]	Shift in
DLE	$10	16	[P]	Data link escape
DC1	$11	17	[Q]	Device control #1
DC2	$12	18	[R]	Device control #2
DC3	$13	19	[S]	Device control #3
DC4	$14	20	[T]	Device control #4
NAK	$15	21	[U]	Negative acknowledge
SYN	$16	22	[V]	Synchronous idle
ETB	$17	23	[W]	End block transmit
CAN	$18	24	[X]	Cancel
EM	$19	25	[Y]	End of medium
SUB	$1A	26	[Z]	Substitute
ESC	$1B	27	[{]	Escape
FS	$1C	28	[|]	Form seperator
GS	$1D	29	[}]	Group seperator
RS	$1E	30	[^]	Range seperator
US	$1F	31	[_]	User seperator
DEL	$7F	127	[DELETE]	Delete

Fig. 3. Standard ASCII control codes can be shown in many different ways.

Names and Numbers

Bar Code News
174 Concord Street
Peterborough, NH 03458
(603) 924-7136

Motorola Semiconductor
Box 20912
Phoenix, AZ 85036
(602) 244-6900

Microswitch
11 West Spring Street
Freeport, IL 61032
(815) 235-6600

Q-SKY Publishing
Box 3042
Springfield, IL 62708
(217) 753-1995

Howard W. Sams
4300 West 62nd Street
Indianapolis, IN 46206
(800) 428-SAMS

Synergetics
Box 809
Thatcher, AZ 85552
(602) 428-4073

↑	is really	[K]	or	VT
↓	is really	[J]	or	LF
→	is really	[U]	or	NAK
←	is really	[H]	or	BS
ESC	is really	[[ ]	or	ESC
TAB	is really	[I]	or	HT
RETURN	is really	[M]	or	CR

Fig. 4. Here are the secret ASCII control codes for the special Apple keys.

Some of the Apple keys are really "hidden" control keys. Figure 4 shows some of these. For instance, the [tab] key is really a [I] HT or horizontal tab command.

Just to thoroughly confuse you, ASCII is a 7-bit code that has only 128 defined states. Since most personal computers work with eight data bits at once, the remaining most-significant bit (MSB) is free for special use by the system designer. This eighth MSB can be used to tell the difference between a key pressed and not pressed, between normal and inverse text on a screen, to mark the end of a word-processing screen line, to tell the difference between Applesloth tokens and real text, or in any of the many other ways the system designer may choose.

Thus, there are two different ASCII codes, depending on who uses the MSB for what. If the MSB is zero, we are using low ASCII. Low ASCII is often used for standard non-Apple text files, and for Apple text files under the ProDOS operating system.

If the MSB is a one, we are using high ASCII. High ASCII is common in older Apples, both for the normal screen display and for DOS 3.3 text files. Since many third-party printer cards expect high ASCII as input, the ProDOS Applewriter 2.0 had to be upgraded to version 2.1 to make sure these cards will not get confused. Note that high ASCII and low ASCII differ by hex $80 or decimal 128. A high ASCII carriage return is a $8D or decimal 141. ME

NEED HELP?

Phone or write your *Hardware Hacker* questions directly to:

Don Lancaster
Synergetics
Box 809
Thatcher, AZ 85552
(602) 428-4073