

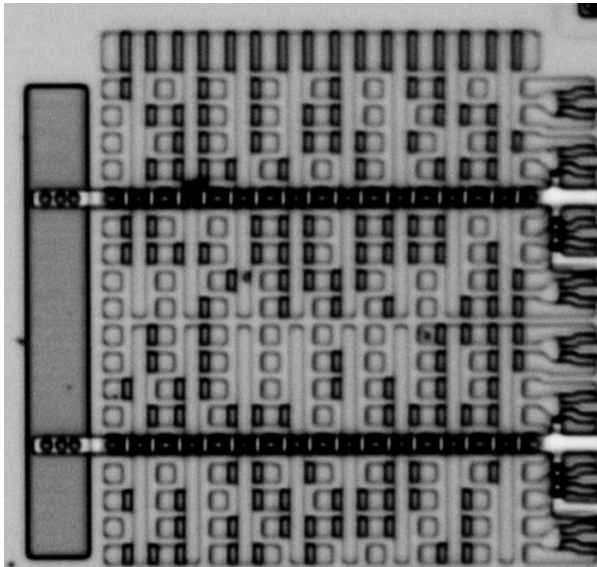
22:02 A Mask ROM Bit Extraction Tool

by Travis Goodspeed

Lately I've been writing a book on extracting firmware from locked microcontrollers; rather, trying to write that book, because I fell into a rabbit hole of mask ROM reverse engineering. So for a few months, instead of writing prose, I wrote a tool in C++ for extracting bits from ROM photographs and a matching tool to decode those bits into logically ordered bytes, suitable for disassembly or emulation.¹

Let's begin with a little background: SRAM, DRAM, Flash ROM, and EPROM memory technologies hold bits invisibly as some form of electrical charge. Mask ROM is different, in that bits are written into one of the lithography masks that produce the chip. This is very expensive per unique program, but very cheap per chip.

Many chips include a nice, orderly grid of bits that contain code or data. Sometimes this is encoded in metal vias, which you can see from the surface of a decapsulated chip. Sometimes bits are in the diffusion layer, and you need to remove the upper layers of the chip with hydrofluoric acid to expose them. And sometimes bits are implanted into the doping difference between P and N silicon, requiring a procedure called a "Dash Etch" to stain a color difference into the bits after exposing them.



Whatever the chemical procedure, the end result from the lab is a panorama photograph of the ROM under high magnification, with bits visible to the naked eye. Like the ones on page 6, you will see that some bits are bright while others are dark. These are our ones and zeroes!

If you are more patient than I, you might type these in manually, reading the ones and zeroes off the page in the same way that as children we typed BASIC program listings out of magazines. Instead, it's nice to let a computer do that work, with a human providing a minimum amount of guidance.

Prior Work

The first of these tools to be published was Rompar by Adam Laurie in 2013.² It's a GUI application in Python, in which the operator draws a grid to mark the bit positions. OpenCV takes care of a bit of image preprocessing, to make the bits stand out by tossing away unneeded color channels.

Published later in 2019, but perhaps written earlier, is Chris Gerlinsky's Bitract.³ The user first loads an image and then describes an "area of interest," a box containing so many rows and columns of bit positions.

These tools certainly work, but they have some problems that frustrated me enough to write something new.

Bitract requires a commercial image processing library to compile in Borland C++. As a Windows program, it ignores command line parameters and has no CLI. As a Python script, Rompar makes too much use of command-line parameters, requiring the row and column count of each bit grouping to be defined before startup rather than worked out on the fly.

Both Rompar and Bitract expect bits to be perfectly ordered in a grid, which is great for reducing the operator's labor, but difficult on very large projects, where camera or stitching distortions might move something just barely out of the grid. It's also inconvenient on some 4-bit microcontrollers, where the final group of bits sometimes has fewer columns than the others.

¹`git clone https://github.com/travisgoodspeed/maskromtool`

²`git clone https://github.com/AdamLaurie/rompar`

³`git clone https://github.com/SiliconAnalysis/bitract`

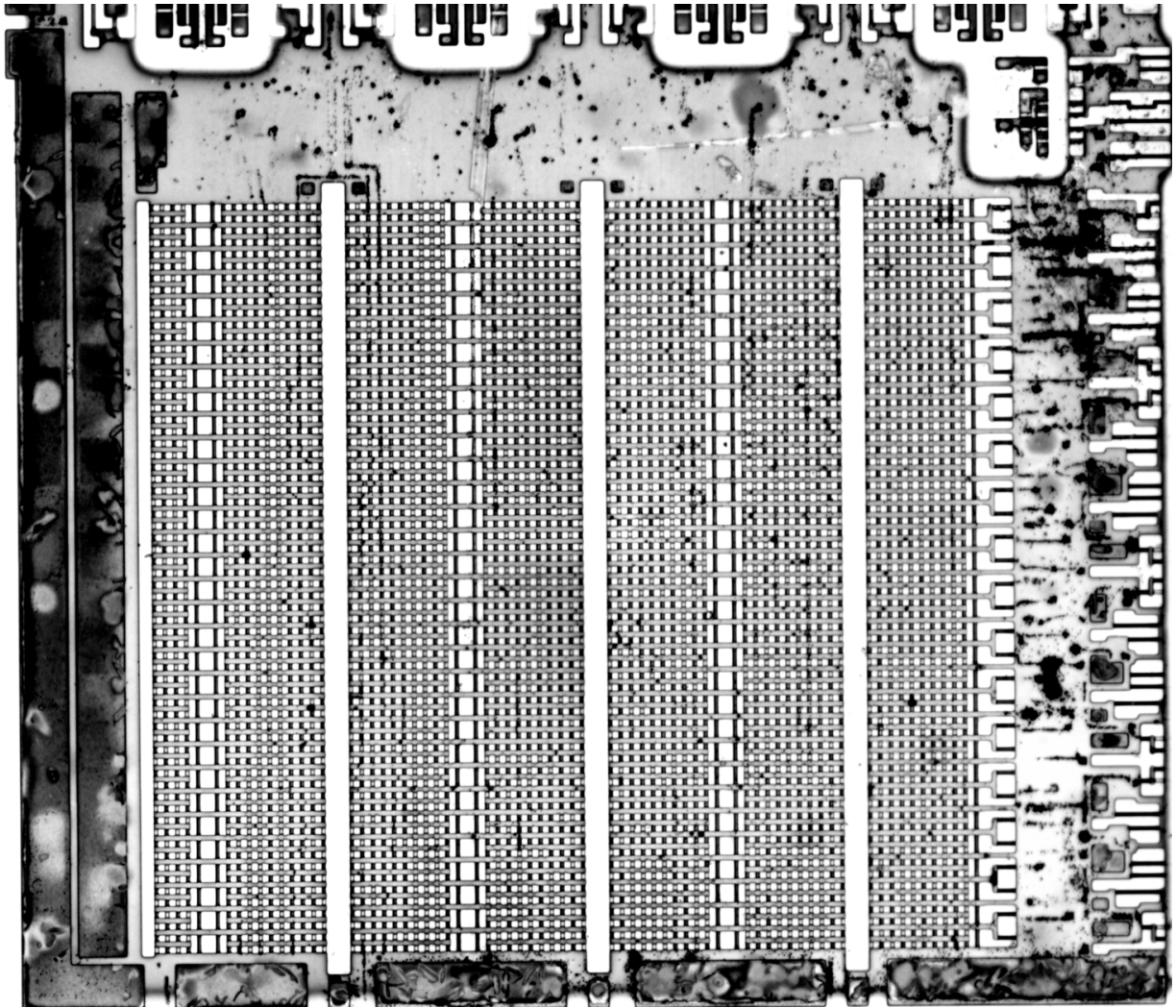


Figure 1: Font ROM from a TMP47C434N

Bitract uses the mouse wheel to zoom, which is infuriating on a multitouch pad that ought to be able to both zoom and pan. Rompar, by contrast, traps the user at the native resolution of the image.

I write these things not to criticize their work. These were damned handy tools for their day. I just think that things could be more convenient.

A Fresh Start

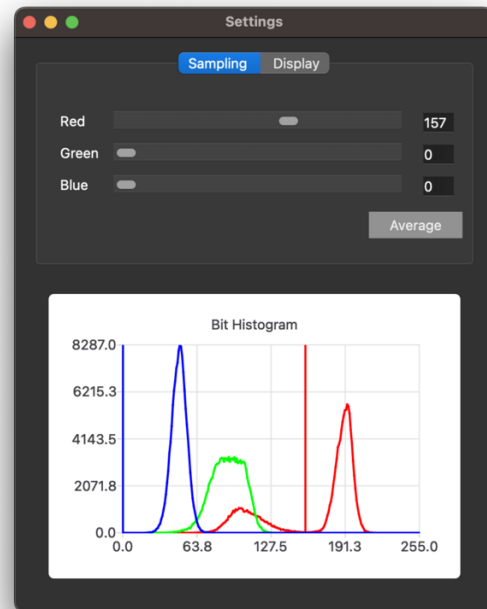
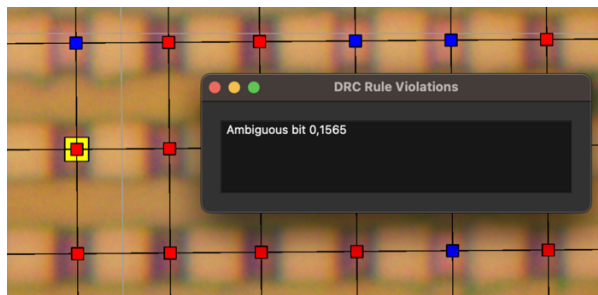
So I began from scratch, with a design on a paper.

For starters, I decided to support both a GUI and a CLI. I wrote the GUI in Qt6, for portability to Linux, Windows and macOS. The CLI is handy for regression testing and scripting, but it is strictly optional. All important features are available without it.

Like Rompar, I chose JSON as a save format. Like Bitract, my tool shows handy histograms to quickly choose the best bit threshold.

I decided to avoid having a strict grid of bits, but instead to use other objects to generate bit positions. In my early drafts, this was done by drawing row and column lines, then identifying their crossing points as bit locations. Because only the bit locations really matter, there's plenty of time to add support for marking grids later.

Since this is a CAD program at heart, I took a few lessons from the PCB layout tools that I regularly cuss at. Design Rule Checks (DRCs) were written early, implementing such features as identifying overlapping bits, ensuring that each row is the same length, and sanity checking the design in other ways. Each DRC violation has a position in the project view, appearing as a yellow box beneath the bits but above the photograph. DRC violations can also be used for providing other feedback; the list isn't necessarily restricted to errors.



Determining a Bit's Value

Knowing the position of a bit, how do we determine whether it's a one or a zero? The short answer is that we can look at how bright or dark it is, but there are some complications to consider.

The first is the color space. Often the bits are distinct in one color channel but absolutely indistinguishable in another. And once we know the right channel, we must select the right threshold to distinguish them.

I found that by drawing a histogram of the number of bits of a given color value, I could quickly see a bimodal distribution in some channel between ones and zeroes. Sliding a channel threshold automatically updates all visible bits, as well as updating a marker in the histogram to visualize where your threshold is set.

I don't use OpenCV or similar libraries to preprocess my images. Rather, I've found that most implant and contact ROMs consistently have a color difference that can be found on a single pixel when working with losslessly compressed images.

Diffusion ROMs are a little different, in that they are low in the chip, and when the chip has been processed a little too long, there's no color difference in the bit's center. Rather, the bit has a dark border. For these and other edge cases, my tool abstracts

away the measurement as a class that returns an RGB triplet. To support this edge case, I simply wrote classes that measured a thin horizontal or vertical strip of pixels, returning the darkest point in each color channel along that strip.

Aligning Bits Into Rows

After the user draws lines for all the rows and columns of the ROM, we take those intersecting points and produce a set of bits positions. Because we don't have a definite grid, it's necessary to *align* these bit objects into rows.

Initially I solved this problem very inefficiently, implementing a function to find the next-to-the-right of any bit by restricting its angle and marking all bits I'd already passed. This worked great for small ROMs, but it scaled horribly, and by one hundred kilobits it was taking twenty minutes to align the bits!

To come up with a faster algorithm, I realized that sorting all of the bits by their X coordinate would *almost* group them into columns. The exceptions come from the image's tilt. Sometimes the leftmost bits of the second column are to the left of the rightmost bits of the first column.

We can therefore identify the leftmost bits by following the sorted list. Whenever the Y gap is small, say less than a few times the average gap, we're still on the first column and we've identified another row header. If the Y gap is large, we're seeing a bit from the second column, and we ought to pass it by. When we start to see many large gaps, we've passed the first column entirely and know all the row header bits.

YOU'LL GO INSANE!!

We've finally released our industry-standard data interfaces for use with the Timex-Sinclair 2000, giving you so vast an array of computer peripheral choices that even trying to make up your mind may push you right over the edge!

CENTRONICS PARALLEL INTERFACE MICROBUILT \$699⁹⁵
The amazing Centronics Parallel Interface lets you connect your TS2000 to just about any dot matrix printer and most other parallel devices. You'll have the option to use the printer's standard font or the TS2000 display font! What's more, we'll throw in printer driver software that supports LPRINT and LLIST absolutely free!

RS232-C SERIAL INTERFACE UNRELEASABLE \$899⁹⁵
Ever fantasized about connecting a letter-quality printer to your TS2000? What about a modem? With the spectacular RS232-C Serial Interface you can do both — at the same time! That's right, we give you two channels of RS232-C power on one board! Bit transfer rate is adjustable from 300 to an incredible 19200 baud. As if that weren't enough, you get free driver software that supports LPRINT and LLIST!

Send your check or money order to:
AERCO Box 18093, Austin, TX 78760
(512) 331-0719

If you're not out of your tree with hardware choices by now, just think about the thousands of software choices you'll have when we release our CPM for the TS2000, scheduled for April 1984. You'll go insane!

So to align the bits, I first build an array of the rightmost bits of each column that I've yet passed. This array is seeded with the row header bits at the far left. I then walk through the sorted list of all remaining bits, overwriting their nearest row element in the array after updating the old bit's `nextto-right` pointer to aim at the new bit.

This is lightning fast, reliably arranging hundreds of thousands of bits in the blink of an eye.

From Physical Bits to Logical Bytes

By this point in the article, you should understand how you might use MaskRomTool to mark the bits of a photograph and arrange them into a table of bit values. You also understand how the DRC mechanism might flag bits which are too near the threshold between a one and a zero. But there's a very important piece we haven't yet covered: How does the software convert this table of physically-ordered bits into logically-ordered bytes?

Let's begin with the prior art. John McMaster's Zorrom tool is built as a set of Python scripts with libraries for CH340, LC5800, LR35902, MCS48, PIC1670, and some TMS320 chips.⁴ For those chips that it doesn't directly support, it has a solver feature that will attempt many permutations of decoding until the bytes match a defined pattern, such as setting the stack pointer in the first instruction. John's solver works for roughly half of targets, and it's far easier than manually guessing permutations. This was the tool that I used until I recently wrote my own decoder.

Chris Gerlinsky's BitViewer uses the a totally different strategy.⁵ Rather than automatically searching permutations, it instead graphically displays the bits with adjustable grouping into columns. This helps a human operator explore the layout, while overdosing on caffeine in a hyper-focused fugue until eventually the bits make sense. This understanding doesn't come easily, but I and others have done it.

I wanted the best of both these worlds. From Zorrom, I wanted a CLI tool that could quickly process my projects, driven by a Makefile to rerun them in order to catch regressions in my decoder or images. I also desperately needed a good search feature, and Zorrom was the only example of such a thing when I started. And from BitViewer, I wanted

⁴ `git clone https://github.com/JohnDMcMaster/zorrom`

⁵ `git clone https://github.com/SiliconAnalysis/bitviewer`

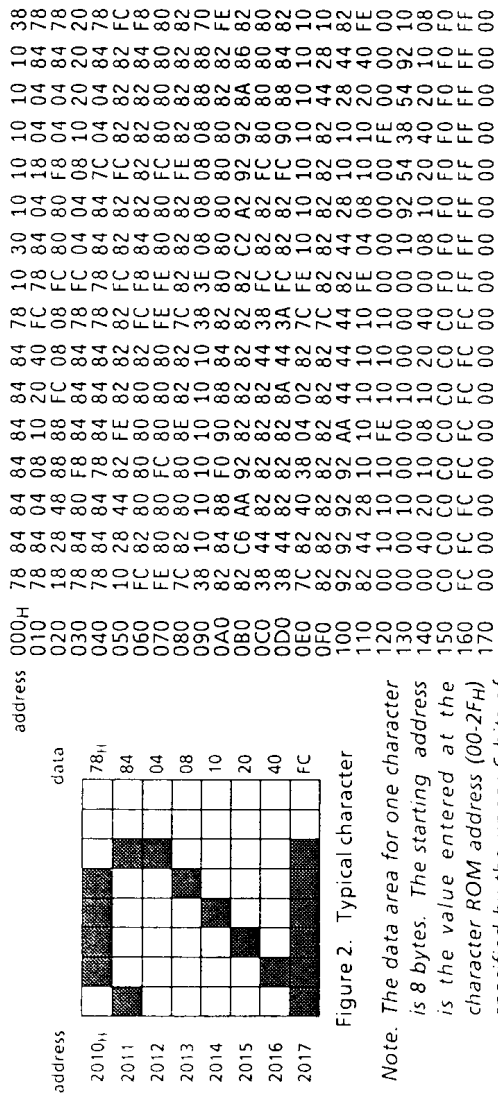
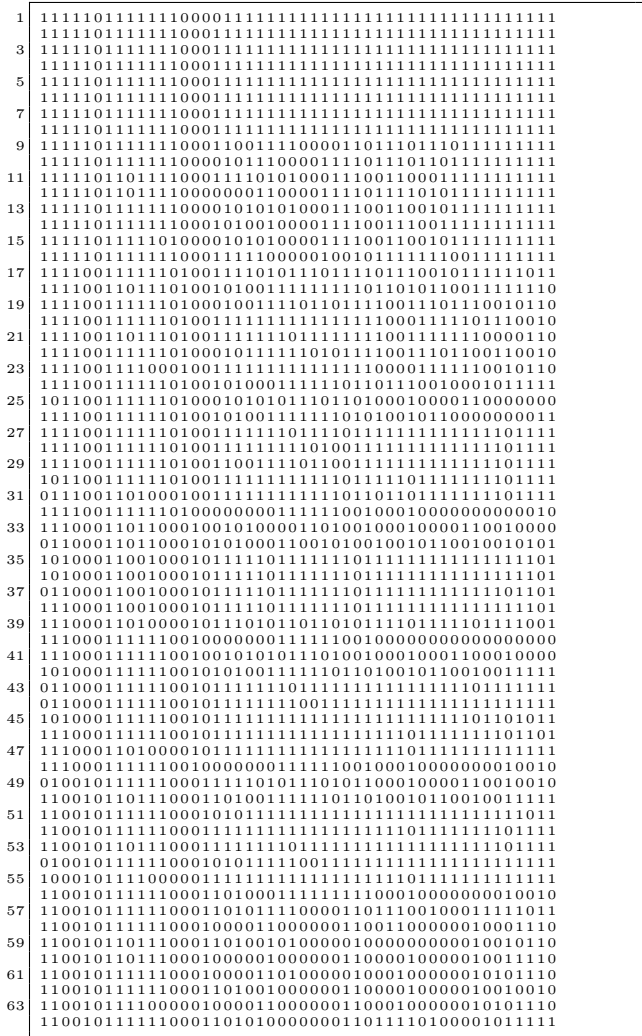


Figure 2. Typical character

Note. The data area for one character is 8 bytes. The starting address is the value entered at the character ROM address (00-2Fh) specified by the upper 6 bits of the 9-bit program area (000-17Fh).

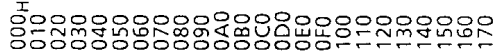


Figure 3. Standard character data (from address 2000H)

Figure 2: Extracted Bitstream and Datasheet Bytes of the TMP47 Font ROM

some graphical connection to my project file, so that a nearly correct guess could be explored until the error was found.

My decoder is called GatoROM. It is used either as a CLI tool without the GUI overhead of MaskRomTool, or as a C++ library within the GUI.

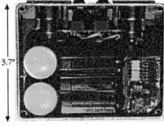
Since Zorrom was the gold standard of solving for unknown layouts, I began my decoder with the complete set of Zorrom permutations from an arbitrary bitstream, essentially exporting every case that it would investigate. Once I could match decodings of all these, and also permute between all settings, I knew that my solver had feature parity with McMaster's.

GatoROM uses its own class to represent a bit, different from that in MaskRomTool. The class holds values such as its address and bitmask during the most recent decoding, as well as a void pointer that might point to a matching bit in the GUI.

All of the GatoROM bits begin in a table of their input positions, and transformations (flip, rotate, etc) produce a table of bits in the output order. This output table is then passed to the parser for decoding, when the address and bitmask fields of the bit class instances are updated to record their logical positions. Zorrom does these steps in roughly the same order, but by passing values instead of pointers, it does not preserve relationships between the inputs and outputs.

I wrote earlier that I also wanted something like BitViewer's interactive nature in my tool. By recording the address and mask of every bit that is decoded, my tools can easily show their work. It's no trouble to select the first few bytes of a decoding, then ask the tool to highlight the bits of those bytes in physical order.

Specifications

<p>± 25 dB 15 Hz - 40 KHz ± 1 dB 5 Hz - 100 KHz < .01% T.H.D. Equivalent input noise < -127 dBm</p>	<p>+16 dBm into 600 Ω 25 Mw into 40 Ω headphones 8-12 hours continuous operation on internal 9 volt batteries depending on phantom load.</p>	
--	---	---

\$ 2 1 0

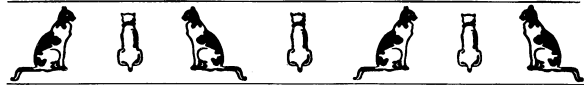
<p>An Excellent Mic Pre-Amp. The AERCO MP-2 is designed with what we believe to be the very best components available.</p> <p>Jensen Input Transformers have no known rival. Exotic materials and financial devotion combine for 5dB points of 2 Hz and 250 KHz.</p> <p>Extremely Low Noise We selected the Linear Technology LT-1028 amplifier on the basis of sonic clarity and super low noise performance that will instantly bring a grin to your face. The fact that it drives +24 dBm into 600 Ω, that it maintains less than 100 uV of D.C. offset at the output, and that it consumes just 200 mW of power are more icing on the cake.</p>	<p>High Efficiency Power converter operating at 500 KHz supplies all the internal power requirements. It has been designed for minimum noise generation and maintains a conversion efficiency well in excess of 90%. Input power can range from 7 - 20 Volts D.C. without regard to polarity and with no electrical connection to the audio circuit.</p> <p>High Gain... Low Gain eight independently selected gain settings for each channel from 20 to 50 dB. Gains are set by a switch and network of precision resistors instead of a pot. This ensures the lowest possible noise and eliminates the reliability problems inherent in small pots. The switches are accessed through the RCA jacks.</p>	<p>Phantom Power is implemented with the industry-standard 52 volt circuit for proper operation of 12 volt and 48 volt microphone types. Individual jumpers for each channel disable the phantom voltage for use in unbalanced environments.</p> <p>Great Quality - Small Price We've priced the 2 channel unit at \$560 to make you a fast friend. Then we plan to sell you more neat stuff.</p>
--	--	---

AERCO
Box 18093 Austin, TX 78760
(512) 451-5874

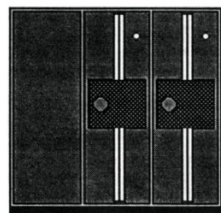
Neighborly greetings to John McMaster for his ground breaking work on Zorrom, for helping me get my lab back together, and for patiently explaining all those things of semiconductor reverse engineering that I had fallen behind on in the past years. And cheers to Vicki Pfau for being smart enough to decode the arrangement of the TMP47 font ROM, which will soon lead to a general decoder for TMP47 program ROMs.

My tools don't yet solve every ROM that was ever manufactured, but I'm happy to say that they are now the best tools for any particular ROM extraction job. They are fast, they hardly ever crash, and they run reliably from the command line or from an OpenGL GUI, whichever you might prefer.

Now that I've solved that problem, perhaps I can get back to finishing my book?




TS 2068 DISC SYSTEM

<p>FD-68 INTERFACE Controls 1-4 drives 3 inch to 8 inch drives Shugart compatible Single or double sided 40/80 tracks per side 64K RAM & 8K ROM on board RGB monitor output</p>		<p>SYSTEM COMPONENTS \$199 FD-68 Interface \$99 Drive 40T/DS/DD 5 inch/400 kilobyte \$99 Dual Drive Cabinet and 5 amp Pwr Pack \$3 Per Item S&H Texas Residents add 5% VISA/MasterCard add 5%</p>
---	--	---

Enhance the performance of your TS 2068 with the AERCO Disc System. All of the speed and convenience of a full-out floppy disc system. Load programs at an incredibly fast 250,000 bits/sec. Fully compatible with all Shugart type drives, including those already in use with the AERCO 1000 Disc System. The 64K of on-board RAM can be used as a second bank of system memory or a soon to be released full-blown CPM System (version 2.2). The RGB output is crystal clear and rock steady. The power supply is a 5-amp high efficiency switcher. We offer a variety of other hardware for all models of SINCLAIR-TIMEX.

	TS/2068	TS/1000-1500
Floppy Disc Interface	\$199	\$179
Disc Drives	from 99	from 99
Power Supplies	99	99
Centronics Printer I/O	69	99
Dual RS-232C Serial I/O	99	99
Direct Video Mod (DV-1)	n/a	15
C ITOH 8510 Printer	375	375
C ITOH 7500 Printer	275	275
ROM Bd. with Auto Disc Boot	n/a	59
RGB Cable (specify monitor)	30	n/a
CP/M (V. 2.2)	coming soon	n/a



Box 18093 Austin TX 78760
Ph (512) 451-5874

