Digital Group Computer

User Manual

Table of Contents

Overview

I had used this Digital Group computer as my main computer MANY years ago when I first started working with computers. I started out with a working computer and, over the years, things broke, or were lost during my many moves. After I retired, it became an item on my bucket list, to re-build the computer and get it working. You can see a history of the project in the link below.

Project History\Digital Group History.pdf

I chose to implement many of the features using PIC processors as they have better development tools available these days then the Z80. Here is the PIC that I chose to use PIC Processor\PIC16F15244.pdf.

Components

Chassis

Here are various links to items used to design the case used to house the Digital Group computer.

Here is a link to the mechanical CAD files I created. This was produced using the software downloaded from ProtoCase.com. They turned out to be very helpful and I would recommend them for any prototype case designs.

Chassis\Protocase Chassis Design.pdf

Here is a link to the card rack I used inside the enclosure.

Chassis\Subracks - VectorPak.pdf

Here are the bus bars I used to simplify the low voltage wiring inside the case.

Chassis\Bus Bar.pdf

Here are the fans that I was planning on using in the enclosure. The mounting patterns are built into the enclosure, but I ended not using them as there was not enough of a heat issue to warrant them.

Chassis\Fan OD4028.pdf

Here is a link to some custom wiring I added to the case to allow the floppy controller to switch 110VAC power on/off to the floppy drive motors. This is covered in more detail in the floppy section.

Chassis\Shugart 801 Power Switch\Shugart 801 floppy power switch.pdf

Power Supply Requirements

The original Digital Group computer used very large linear power supplies that produced lots of heat. Rather than reproduce this inefficient design, I opted to replace it with three separate switching power supplies as follows:

1. +5VDC @12A
2. +5VDC @4A, +12VDC @2A, -12VDC @0.5A
3. +24VDC @4.5A

The first supply runs all of the logic. I initially was going to use the 4A on the second supply but it turned out to be insufficient when powering all of the boards.

The second supply is only used for the +12/-12VDC requirements, the +5VDC is not used on this supply.

The third supply is used only for the 8” floppy drives which require this for the track stepper motor drive.

CPU Motherboard

The CPU motherboard connects all of the Digital Group CPU cards together and provides the following additional features:

1. I/O port address decoding for all boards and expansion board
2. A port mapped DIP switch used by the boot ROM to enable SSD boot and debug breakpoints
3. Support for up to six of my custom expansion cards (described later in the document)
4. An expansion connector that links each expansion board to the outside world

I went through several versions of the motherboard. Original versions were extremely unstable as they had no power planes which turned out to be essential.

Here is a link to the latest version of the motherboard.

CPU Motherboard\76000430 CPU Motherboard.pdf

Tragically, I made one mistake on this board but it wasn't worth re-doing the whole board. The issue is that the 6x expansion connectors don't have the WAIT line hooked up to pin 6, instead it is connected to ground. Because ground is a power plane, you can't cut the trace, instead you need to bend pin 6 of the connector up (so that it doesn't go into the hole) when soldering the connector. Then run a wire-wrap wire through the hole and connect it to the bent up pin. Run the other end of the wire to the CPU WAIT line.

Here is a document for the 72 pin connectors needed.

CPU Motherboard\S100 72pin Connector.pdf

The 100 pin S-100 connector is no longer available. I was able to find from a guy in Bulgaria on EBay. Here is the contact information I have.

lzet_electronics on eBay

100 pin 2x50 S100 Bus Backplane Pitch Card Edge Connector MITS Altair Imsai | eBay

Zhori LZ Electronics lz_electronics@mail.ru

CPU Card

The CPU card originally had provisions for 2K of static RAM. This is no longer used – instead an external memory card is used. The CPU card does almost nothing except provide support for a BOOT ROM and bus interface.

Here is the original CPU card user manual. **IMPORTANT** – note that there is a mistake on the schematic incorrectly showing "Data To Memory" edge connector pins 20-13 as D7-D0, when it is actually D0-D7.

Digital Group CPU Card\Z80 CPU Card.pdf

Note that the original wait state logic would only provide a 'brief' wait pulse for any off-board wait request. This had to be modified to provide an 'indefinite' wait pulse for the motherboard expansion boards. This hand made circuit is described below.

Expansion Cards\Wait State Circuit Modification\Z80 CPU - Wait Circuit Fix.pdf

Here are various other hard to find IC documents.

Digital Group CPU Card\Z80 Users Manual.pdf

Digital Group CPU Card\2112A 256x4 Static RAM.pdf

Digital Group CPU Card\MM5290 RAM - can't find MM5275.pdf

Digital Group CPU Card\8T97.pdf

Digital Group CPU Card\DM9602.pdf

Digital Group CPU Card\DP8224.pdf

Here are my notes about diagnosing a particularly hard to find startup issue.

Digital Group CPU Card\Startup Failure State.pdf

Digital Group CPU Card\Startup Success State.pdf

EPROM Adapter Card

This is a very straight forward card that allows larger EPROMs to be interfaced to the archaic Intel 1702 EPROM socket.

EPROM expansion board\76000530 EPROM Adapter.pdf

EPROM expansion board\1702 EPROM.pdf

EPROM expansion board\2716.pdf

Boot ROM

The Boot ROM is responsible for powering up the hardware and providing Z80 program support for the keyboard and display.

DIP Switch Settings

- SW1 – 'open' = boot from SSD (if present)
- SW2 – 'open' ignore Boot ROM breakpoints
- SW3-8 – not currently implmented

LED Status Indicators

Used to display the state of the Boot ROM during power up and can be used to trouble shoot hardware issues if necessary. See Boot ROM source code for their current meaning.

Breakpoints

While trying to get the Boot ROM working, I ran into many issues. With no hardware debugger available, I was forced to implement a poor man's breakpoint that could be placed into the Boot ROM source code at various locations.

Basically it operates by checking to see if SW2 is closed (enabling breakpoints), if so the logic 'spins' at the current program location until the user manually sets memory location 0x8000 to 0x00. This can be accomplished using the memory 'set' functionality built into the Front Panel serial port commands.

Using this crude device, I was able to identify many programming errors that otherwise would have been impossible to identify.

Now that the Boot ROM is working, this feature is of little use, but may come in handy if new features are added to the Boot ROM in the future.

Commands Supported

The following commands are implemented using the keyboard and monitor through the Boot ROM (double quotes are used to delimit the command – do not type them into the keyboard.)

- “D ssss eeee” – dump memory from hex address ‘ssss’ to ‘eeee’
- “F ssss eeee dd” – fill memory with hex value ‘dd’ from hex address ‘ssss’ to ‘eeee’
- “I pp” – read port at hex address ‘pp’ and display the value
- “O pp dd” – write hex value ‘dd’ to hex port address ‘pp’
- “C” – toggle cursor on/off
- “R ssss” – run code at hex address ‘ssss’
- “S ssss” – copy 16K of memory starting at hex address ‘ssss’ to the SSD boot area, note that when the code is loaded back into RAM at boot time, it will be copied back to the same address ‘ssss’ before it is run.
- “T” – detect working RAM and run extensive walking bit tests on the RAM to determine any bad areas. These tests run forever and keep a running tally of all good/bad RAM found. You must reboot to exit this test.

Tools Required

The assembler tools I found from this page.

http://www.z80.info/z80sdt.htm

Specifically, I ended up using ZCC compiler package (from this page also) which can be directly downloaded here.

http://www.z80.info/zip/zcc096.zip

This has many downfalls, but does work. Some limitations are the linker will only link a few files – go over about 10 or so and it stops working. But it comes with source code so you can fix it.

The executables that come with it will NOT work on a 64bit PC. I originally setup a Windows XP virtual machine to run the tools (which was a pain to do) but it later dawned on me that I could re-build the executables in Visual Studio. This was relatively simple once I looked into it. I've listed the issues below, otherwise it is pretty straightforward.

1. Need to lower the compiler warning level to '2', otherwise it flags all the 'old' style function definitions and safety violations.
2. Need to modify a bunch of files that include <alloc.h> (the old Borland version) to use <malloc.h>

That's it! Now you can use the tools on 64bit Windows.

Here are links to the Boot ROM source files (to download the original source file see appendix.)

Boot ROM\BreakPoint.pdf

Boot ROM\CmdLine.pdf

Boot ROM\DispHLDE.pdf

Boot ROM\DispUtil.pdf

Boot ROM\HardDisk.pdf

Boot ROM\Hardware.pdf

Boot ROM\io.pdf

Boot ROM\MainLoop.pdf

Boot ROM\Monitor.pdf

Boot ROM\OSIF.pdf

Boot ROM\OSIFCode.pdf

Boot ROM\PowerUp.pdf

Boot ROM\RAMTest.pdf

Here is a link to the DOS batch file used to invoke the Z80 tools to build the Boot ROM.

Boot ROM\Build Batch File.pdf

Original Digital Group 8K Memory Card

This is a pretty standard RAM card and all the details are covered in the document below.

Digital Group 8K Memory Card\8k Memory PCB DG-0011-A.pdf

This was the original Digital Group RAM card and only had 7K populated on my version. I think the other 1K was removed and used to populate the video card RAM.

I opted to use the Godbout 32K RAM card instead because, well it's bigger, and I suspected there was some unreliability issues with the Digital Group card and didn't want to spend the extra time tracking it down.

Godbout 32K Memory Card

This is a pretty standard RAM card and all the details are covered in the document below.

Godbout 32K Memory Card\Godbout_Econoram_IX_Feb79.pdf

Video Card

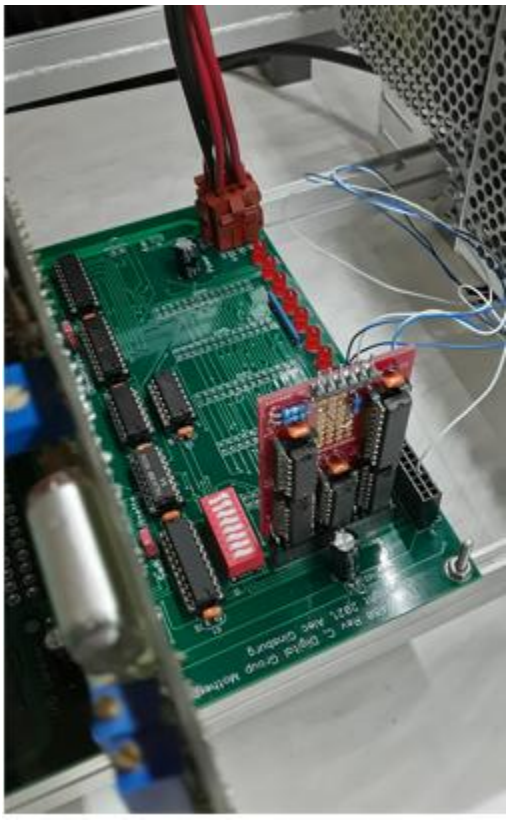
The video card is describe pretty well in the original Digital Group document below.

Digital Group Video Card\TV & Cassette Card.pdf

This is a very low resolution display with very few features. It was a challenge to make it behave like a basic terminal. All of the code to do so is located in the Boot ROM.

Expansion Bus

The expansion bus supports up to six PIC based expansion cards that can be accessed from the Z80 CPU bus at fixed I/O port locations. The picture below shows a single expansion card installed.



The Digital Group CPU only supports a 'momentary' WAIT request from peripherals. It is therefore necessary to modify the Digital Group CPU as shown in the link below:

Expansion Cards\Wait State Circuit Modification\Z80 CPU - Wait Circuit Fix.pdf

The following link shows that generic expansion card. Specific expansion features use this card with extra circuitry hand added in the prototyping area.

Expansion Cards\Generic\76000830 Expansion.pdf

Keyboard Expansion Card

The keyboard expansion card provides the interface between a standard PS/2 keyboard and the Digital Group Z80 CPU board.

The interface is surprisingly complex and would likely exceed the capabilities of the Z80 system (especially since it has no synchronous serial ports.) The following gives a good overview of the interface requirements:

Expansion Cards\Keyboard Interface\PS2 Keyboard.pdf

Here is a link to the keyboard schematic. Note that this is NOT a custom board, but requires the user to hand wire several components in the prototyping area of the standard expansion PCB.

Expansion Cards\Keyboard Interface\76000830 Expansion - Keyboard.pdf

Here is a link to the source code required for this interface (to download the original source file see appendix.)

Expansion Cards\Keyboard Interface\PIC Source Code.pdf

Solid State Disk Expansion Card

The SSD expansion card provides a hard disk interface to the Z80 bus.

It currently has sector read/write support as well as a 'bootable' area that can be used to boot the Z80 on power up to run any given 16K piece of code. This area is initialized by:

1. downloading the code into the correct memory location using the front panel serial interface
2. copying the code from RAM into the boot area using the front panel 'S'ys command

The boot ROM code currently looks for this boot area on power up and will load and boot the code if DIP switch 1 is in the 'open' state on the motherboard. To disable the automatic boot, change DIP switch '2' to the 'closed' state.

Here is the schematic for the SSD. Note that this is NOT a custom board, but requires the user to hand wire several components in the prototyping area of the standard expansion PCB.

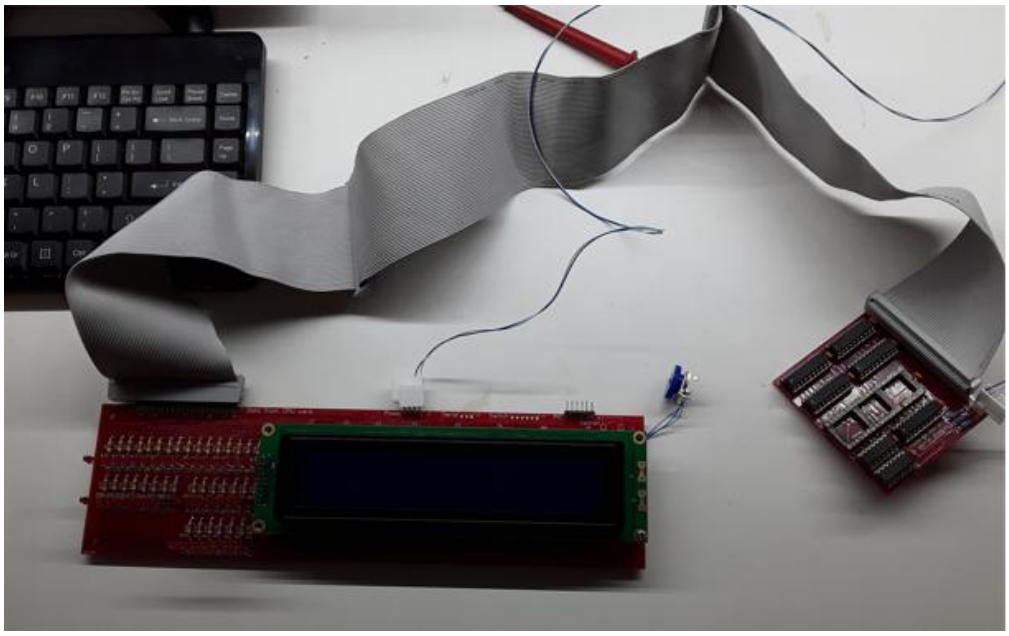
Expansion Cards\SSD Interface\76000830 Expansion - SSD.pdf

Here is a link to the source code for the PIC processor on this board (to download the original source file see appendix.)

Expansion Cards\SSD Interface\PIC Source Code.pdf

Front Panel Assembly

Front Panel Display (left) and Front Panel Interface (right) connected by ribbon cable.



Front Panel operating in chassis. Note Reset/Single Step/Run-Stop switches near floppy.

The front panel implements a feature similar to the old IMSAI 8080 wherein the user can observe the CPU state on a set of LEDs.

It extends the IMSAI functionality by decoding the processor state on an LCD display.

The IMSAI toggle switches were omitted, partially because they are no longer commercially available, and partially because this form of data input is incredibly tedious. Instead a monitor feature was implemented over a serial port that allows the user to perform similar functions much more easily.

The Run/Stop, and Single Step features are implemented using three front panel switches. These features however, are not part of the front panel – they are supported by the Digital Group CPU card.

Front Panel Interface

Front Panel Display\CPU Interface Board\76000730 Front Panel Interface.pdf

The Front Panel Interface board requires, and works in conjunction with, the Front Panel Display board.

The Front Panel Interface is installed by:

- Removing the Z80 processor chip on the CPU board.
- Placing the Z80 processor chip into the CPU socket on the Front Panel Interface
- Connect the Front Panel Interface board with the Front Panel Display board using the appropriate ribbon cable.
- Inserting the Front Panel Interface assembly into the Z80 socket on the CPU board

The front panel logic works by taking over the Z80 processor bus using the Bus Request feature of the Z80.

Front Panel Display

The Front Panel Display provides the archaic LED display of the Z80 bus state, as well as an updated LCD display that actually decodes the bus state into easily readable text.

Front Panel Display\Front Panel Board\76000630 Front Panel.pdf

The Front Panel Display board requires, and works in conjunction with, the Front Panel Interface board.

Components

Front Panel Display\Front Panel Board\4x40 LCD.pdf

Front Panel Display\Front Panel Board\4x40 LCD Controller.pdf

Front Panel Display\Front Panel Board\ASS_4888_CO.pdf

Monitor Feature

Interface Requirements

Serial interface: 9600 baud, 8 bit, 1 stop bit, no flow control

Minimum 200mS delay between sending packets to allow the firmware to process the last packet. This is especially crucial when downloading an Intel hex file.

Commands Supported

The following commands are supported (unless otherwise specified, all values are in hex):

- I pp – read value on port 'pp'
  - Returns "Port = xx" where 'xx' is the value of the port
- O pp dd – output data 'dd' to port 'pp'
  - Returns "OK" if successful.
- D ssss:eeee – dump memory from 'ssss' to 'eeee'
  - Returns "Mem ssss = dd dd … dd" where 'dd' is the value at successive memory locations.
  - **IMPORTANT** – there is a 40 byte limit on the output string so only a few bytes can be dumped at a time. Dumping large blocks will result in only the first few bytes displayed and the rest truncated.
- F ssss:eeee dd – fill memory from 'ssss' to 'eeee' with value 'dd'
  - Returns "Ok" if successful
- :Intel hex record – load data specified by Intel hex record into memory
  - Only valid hex records will be processed
    - Record must start with ':'
    - Record must be at least 11 bytes long
    - Record must have the following format
      - ':',
      - byte count (2 bytes),
      - address (4 bytes),
      - record type (2 bytes)
      - optional data – specified by byte count
      - checksum (2 bytes)
  - No status is returned for valid records
  - Invalid records will return "Bad Cmd"

Any un-supported commands will return the following over the serial port:

"Bad Cmd"

Floppy Drive Motherboard

The purpose of the floppy drive motherboard is to offload the CPU burden required to operate the floppy interface from the main CPU, and to make it easier to debug the floppy interface using up to date diagnostic tools for a PIC process (rather than the Z80 which has almost no support today.)

The floppy drive motherboard is a separate PCB from the main Digital Group motherboard for two reasons. First I wanted to get the main Z80 boards working quickly, without getting mired down in the more complicated task of implementing a floppy interface. And secondly, the floppy interface board requires 2 connectors, and is therefore much wider than the CPU motherboard. If the floppy functionality was placed on the main motherboard, it would have required a much larger, and more costly, motherboard.

The schematic for the motherboard is here:

Floppy Drive Motherboard\76001030 FDC Motherboard.pdf

One shortcoming of the Digital Group floppy interface is that it did not provide any means of turning the floppy drive motor off. As a result the floppy drive motor continues to run constantly, making a lot of noise and needlessly wearing the motor, belts, bearings, and etc. out prematurely.

To overcome this I designed a simple circuit using digital relays connected to the drive select lines of each floppy drive to enable the drive motor power.

Chassis\Shugart 801 Power Switch\Shugart 801 floppy power switch.pdf

The floppy interface is over a single serial port with either TTL or RS232 levels. The intent was to have the TTL interface wired up to one of the Z80 expansion cards that would provide the interface between the Z80 and the floppy interface. Optionally, the board could be directly connected to a standard PC using the RS232 interface for debugging.

Note that in order to use the TTL interface, the RS232 driver chip must be removed from its socket.

As it turned out, I never got around to implementing the TTL interface as it turned out to be more fun to use the PC interface to rip data from old floppies that I had laying around for 40 years and see the old files that I used to work with once again.

I had to write a simple PC program (VS 2010) to do this. The source for it can be found here:

Floppy Drive Motherboard\FloppyIF Source Code\main.cpp

One thing to note about this setup is that the PIC firmware will force the entire sector of read data to the same value (see PIC code READ_WRITE_STATUS_TYPE for a full list) if a read error occurs. So if you read a sector and get all 0x05 values this means the sector failed the CRC test and likely is a bad sector with physical damage.

It was while trying to read old floppies that I realized that the floppy controller didn't support double density encoding. This was confusing at first, as I was able to able to read track zero of several of my old double density floppies. After some research I discovered that it was common practice in some circles at the time to encode track zero as single density, and the rest of the disk as double density, so that the controller logic could always read track zero as single density, and determine from its data how the rest of the disk was encoded. A good reference to this can be found here:

Floppy Disk Formats, Standards and Geometry (nj7p.org)

The source code for the PIC processor can be found below (to download the original source file see appendix.)

Floppy Drive Motherboard\PIC Source Code\PIC Source Code.pdf

Note that this code is very time critical – even with a PIC running at 32MHz. It is frankly amazing that this could run at all on a Z80. No interrupts are used, and communication with the serial port must stop while the floppy hardware is being accessed.

Originally, I had to set the PIC compiler optimization level to '3' (very high) to get it working, but was able in the end to get it to work with level '0' (none) which made debugging much easier.

A shortcut was taken in the design in order to simplify the electronics. There were not enough pins left over on the PIC to implement a serial port after the floppy interface was fully implemented and debugged. To overcome this problem, without adding a lot of external hardware, a compromise was reached to 'share' PORTC between the floppy data bus, and the serial Tx and Rx pins, one of the PORTB pins is used to switch between the two states.

This multiplexed bus design works well in this application because of two facts:

1. The floppy interface is so CPU intensive that nothing else can occur while it is in process
2. The host (talking over the serial port) must wait for the floppy response before proceeding

The following are things that could be improved (if I ever get the motivation):

1. The logic to detect if the drive motor is up to speed didn't work and is bypassed
2. The serial interface is only 9600 baud and kind of slow
3. The RemapPins.h file contains defines that should be in compiler files somewhere, but I never found them.
4. Possibly implement re-tries on the read sector logic
5. Make FloppyIF.exe configurable – currently it is hard coded to use COM4, 9600baud, and store the 'rip'ed file as 'RIP.DAT'.

Digital Group Floppy Interface Card

This card was state of the art when it was designed in the late 1970's. It was a revolutionary step forward in data storage compared to the paper tape and cassette tape storage used prior to that.

This board supported up to four, single sided, soft sector, single density IBM (and an older 'half' density Shugart) 8" drives.

As can be expected the NEC uPD372 controller chip, at the heart of the board, had its hands full dealing with the floppy data stream and control features. Therefore there is a heavy burden on the main CPU to handle many of the floppy interface control tasks.

The original Digital Group computer did not have enough CPU overhead to handle the speed at which data came out of this card during reads, even using interrupts. The original driver was written to take advantage of a clever trick whereby the Z80 would enter a 'halt' state which would then be 'released' by the floppy interface card's interrupt line when the next character was available. This required 100% of the CPU resources while accessing the floppy drive.

Originally, I was unable to find any schematics for the floppy interface card, and had to reverse engineer the board to get a schematic. This was very time consuming, and only months later did I happen to find a copy of the user manual on the internet with a schematic.

There are two interesting aspects of the circuit as described below.

First off, there is a complex state machine that is used to satisfy some arcane requirements of the uPD372 chip. I have documented this in detail on my version of the schematic. At first glance this circuit makes no sense and is very confusing until you discover the underlying requirements.

Secondly, the floppy read data logic has a 'data conditioner' which is used to separate the 'data' and 'clock' signals coming from the floppy on a single wire. The 'data' portion is what you would expect. The clock portion allows the floppy controller to differentiate between three different 'types' of data as follows:

1. Index mark – used at the start of the track
2. ID mark – used at the start of each sector block
3. Data mark – used at the start of each data block (following a sector block)
4. Deleted data mark – used to identify a data block that has been deleted

There is an excellent description of how this works in the uPD372 data sheet below, as well as example source code.

Digital Group Floppy Card\uPD372_Floppy_Disk_Controller_Mar77.pdf

Here is a link to my schematic and notes:

Digital Group Floppy Card\DG-0020 Floppy Controller Schematic.pdf

Here is the original Digital Group user manual:

Digital Group Floppy Card\Digitial Group floppy_disk_controller manual.PDF

Shugart 801 Floppy Drive

Here is the original Shugart service manual for the 8” drives that are used:

Shugart 801 Floppy Drive\39025-1_SA800_801_Service_Manual_Mar82.pdf

Here are other (less useful) miscellaneous documents relating to the Shugart drive:

Shugart 801 Floppy Drive\Other Misc Documents\50664-0_SA800_801_Theory_of_Operations_Apr76.pdf

Shugart 801 Floppy Drive\Other Misc Documents\50664-1_SA800_TheorOp_May78.pdf

Shugart 801 Floppy Drive\Other Misc Documents\50574-4_SA800_OEM_May80.pdf

Shugart 801 Floppy Drive\Other Misc Documents\50575-4_SA800_Maint_Feb78.pdf

Shugart 801 Floppy Drive\Other Misc Documents\50576-4_SA800_Parts_May77.pdf

Appendix

Source Code and Gerber Files

- 760004xx S100 Mother Board PCB
  - Gerber Files
- 760005xx 1702A EPROM Adapter PCB
  - Gerber Files
- 760006xx Front Panel PCB
  - Gerber Files
  - Source Code
- 760007xx Front Panel Interface PCB
  - Gerber Files
- 760008xx Expansion PCB (used for keyboard & SSD)
  - Gerber Files
  - Keyboard Source Code
  - SSD Source Code
- 760010xx Floppy Disk Controller PCB
  - Gerber Files
  - Source Code
- Z80 Source Code
  - BootROM
  - Tic-Tac-Toe Program