A Brief Introduction to *Glenside's* IDE adapter

April 30, 1999

About this document

The Glenside Color Computer Club (GCCC) would like to thank Brett Heath for his generous help in writing documentation for the CoCo IDE project. This documentation would not be as comprehensive as it is without his help. We accept responsibility for any errors in fact (or even fiction) that appear in this documentation; errors are by no means the fault of Brett or any other authors. While Brett wrote the bulk of this document – especially the technical IDE chapters – Eddie Kuns and Carl Boll have edited and added to his work. If any errors have crept into this document or remain, we are responsible. (By the way, we are not loath to accept corrections. :-)

This document is by no means comprehensive, it is a general description of the basic IDE interface as defined in ata-r4c.txt. Some reference has been made to the ata3r4 (ATA-3 Release 4) draft standard so that the various tables will be up to date, but note that the enhanced features of the later ATA incarnation are for the most part ignored or given only cursory treatment. While it is our intention to provide a usable synopsis of IDE operation we are not IDE experts. Do not write code based on this document without consulting an authoritative source first!

Contents

Chapter 1

Introduction

Computers, Controllers, and Drives... some background. As anyone who has ever tried to actually use a computer to accomplish something useful knows, computers are incredibly stupid. It matters little whether you are a first time user trying to browse the "Documentation" or a cynical guru writing yet another "User Friendly Interface" . Unless you know exactly what you want to do, and can translate that into a command or specific sequence of commands that the computer already understands, you're likely to be very frustrated.

Less well known is the analogous situation that exists between a computer and its peripherals. In order to read from a disk drive (for example :-) the host must have some means of specifying (identifying) the information required, and also a method of retrieving that particular data in order to use it. For disk systems, identification has traditionally been accomplished by describing the physical location on the disk of the desired data.

Data on magnetic disks is organized in concentric rings (called tracks or cylinders) with each ring subdivided into data blocks (called sectors). A typical hard drive consists of a stack of magnetically coated disks each side of which has its own read/write head, so the host can uniquely identify any data block on the drive by specifying the head, the track, and the sector where it is located.

With early controllers the process of actually accessing the specified data was quite involved. Fortunately however, the IDE interface has largely automated this process so that all the host has to do is set any necessary parameters and issue the command. The drive's electronics take care of positioning the heads, selecting the appropriate head, and reading or writing the chosen sector(s), it then notifies the host if a data transfer is required.

1.1 What It Does

The Glenside IDE card provides an interface between the CoCo 40-pin cartridge edge connector and a 40-pin IDE ribbon cable connector. This means that with appropriate software drivers your CoCo can use up to two IDE devices – in particular the widely available and scandalously cheap IDE hard drives that are made for IBM PC's and their clones.

1.2 A Simplified History (Where did it come from?)

Back at the dawn of civilization when we were younger and the 6809 was new, disk drives were big, expensive, dumb, and very much an extra cost add-on for anybody in the home computer market; hard disks quadruply so.

Since the drive interfaces were somewhat more standardized than the computer architectures (vast understatement:), adding a drive involved finding a controller that would work with your computer and then choosing a compatible drive. The controller would plug in to whatever expansion bus was used on your system and provided connector(s) to whichever 'standard' drive cable(s) it was designed for.

On the circuit board in between would be logic which took care of various mundane control tasks, like remembering the position of the heads, sending the right sequence of step commands to get the heads from

wherever they were to (for example) track 31, letting the system know that the heads are now in the right place etc...

The IBM PC followed this model, a semi-intelligent plug-in card that presented a command/status interface to the processor and a control/timing interface to the drive. With the reduction of diversity that followed the success of the IBM PC came a corresponding motivation to optimize the 'standard' hard drive interface to the IBM PC. Over the years a rough consensus of how to do this was reached, it involved offloading as much housekeeping as possible from the host ('Intelligent Drive Electronics') and/or reducing the number of subsystems by combining controller and drive into one package ('Integrated Drive Electronics'); depending on who you asked. In any case it was known as the IDE interface.

In 1988 a group of manufacturers got together (as The Common Access Method Committee) and (amongst other things) proposed a formal standard to describe the IDE interface. This standard was eventually named the 'AT Attachment' interface (designated ATA) and it specified a logical and electrical connection between the IBM PC/AT expansion bus and local bulk storage devices (especially 'hard' or 'fixed' disks).

Drive technology continued to improve and in recent years the ATA-2 (EIDE) and ATA-3 standards have provided enhanced functionality and performance for the basic IDE interface, particularly by supporting a wider range of devices (including ATAPI CD-ROM), faster data transfers, and larger drives. As an aside, ATAPI stands for ATA Peripheral Interface and this standard describes how to interface with non-fixed-disk peripherals such as tape drives and CD-ROM drives. Such peripherals are not supported by the current driver, but in theory *could* be supported if someone decided to put in the effort to modify the driver.

The current draft standard is labelled ATA-3r7b (ATA-3 release 7b) and it may be retrieved from ftp://fission.dt.wdc.com/pub/standards/x3t13.html.

1.3 IDE: What is it for?

"IDE" is commonly used to refer to what is more properly known as the ATA Interface, after the standard which defines its operation (In this document "ATA" and "IDE" are used interchangeably). The ATA Interface has become the de-facto standard way of connecting fixed disks to AT class personal computers. Although there have been many enhancements since the initial standard, and the current IDE supports more features and a wider variety of devices, the basic hard drive interface remains essentially unchanged. This document therefore will focus on the 'plain vanilla' IDE as described in the draft document ata-r4c.txt. The 'IDE standard' is actually a collection of several distinct but interrelated specifications, which may be grouped into three categories:

- The electrical specifications deal with mundane essentials like I/O timing, cable termination, cable length, etc. Most of these issues are beyond the scope of this introduction, so I'm gonna ignore 'em :-)
- The physical specifications take care of mechanical details such as pin assignments for the interface and power cables, connector types, connector keying to prevent plugging the cable in backwards, and so forth. Since there is an intimate relationship between the signal definitions and the logical interface there are several aspects of the physical interface that will be described in some detail.
- The logical specifications (which are our primary concern here) provide means for organizing and controlling communication between the host and the drive(s).

The purpose of this document is to give an understandable explanation of the logical interface.

1.4 CoCo IDE: Where Did It Come From?

Many of you gentle readers (or not so gentle readers) may remember the discussions that took place on the CoCo List (coco@pucc.princeton.edu) about interfacing an IDE drive to the CoCo. At the time of the discussion, some believed this to be an impossible task! (We've heard that before, haven't we?) Perhaps spurred on by this belief, perhaps merely inspired by the abundance and price of cheap IDE drives (cheap compared – at that time – to any alternative hard disk) Jim Hathaway III decided to attempt to build such an interface.

To make a short story long, he succeeded. Be assured that there was a lot of hard thought, trial and error, and in the end, clever hardware design, but that story is for another time and for another person to tell. The authors have witnessed a series of hardware experts look at the design: "That can't possibly work." (pause) "Oh!" (pause) "No!?" (pause) "Oh my G-d, he's a genius!" We hope you will have a similar experience. :-)

To explain exactly how the GCCC came to be involved, Jim announced on the CoCo List that he had designed, built, and tested a working prototype of an IDE board for the CoCo. Note that this includes writing a functioning device driver – pretty good work for someone who doesn't describe himself as a kernel guru!

When he made this announcement, Carl Boll of the GCCC contacted him about the possibility of the GCCC mass-producing the IDE interface for the entire community. Jim didn't have the resources or time to support mass use of his interface and was happy to allow the GCCC to produce them for the community.

This is entirely an non-profit venture. Any money left over after paying all costs associated with the project (boards, printing, mailing, and support) will be rolled over into a fund to do research and development on new CoCo projects. One possible future project is to produce the Fast-232 serial port. We will be happy to entertain your thoughts, but probably the best arena for this discussion is the CoCo mailing list.

1.5 How Did We Get Here From There?

First, Carl asked Jim if Jim would build an interface and ship it to him. This was to ensure that we had a known-working interface board along *with* the schematics, so when we built our first prototype boards, any errors in our construction could be compared to a working card. (Not that we were paranoid or anything. :-)

After Carl received the interface card Jim built for him, along with a working 8-bit driver and a .GIF file schematic, Carl showed off Jim's work at a GCCC meeting. Yes, this was the meeting at which we heard more than two hardware experts "ooo" and "ahhh" over the schematic.

Brian Schubring then took the .GIF file and made a more literal schematic that could then be used in a PCB layout program. This is the point at which Gene Brooks took the ball and ran for a touchdown. Not only did he turn this schematic into a full double-sided PCB layout (plated through holes, ground plane, and all) but he also used this layout to produce several prototype boards before we sent the layout to a board maker. This was a life saver, because indeed there were a few minor hardware "bugs" in our original layout. The last change made to the layout was one suggested by Carl Boll and Mark Farrell – the ability to choose the hardware address (either $FF7x or $FF5x) via jumper. Mark was first to discover the high $FF7x address conflict with the Multipak control addresses!

Jim's original driver was an 8-bit driver. What does this mean? The IDE spec describes a 16-bit data bus. That is, 16 bits of data are transferred to the drive or from the drive at one time. Of course, the CoCo that we all know and love has an 8-bit data bus! Jim simply used eight of the sixteen bits, letting the other eight float. This means he used 1/2 of the total IDE disk storage capacity, but hey, IDE drives are now available cheaply and he did the project for fun anyway. Carl, Eddie and Mark wished to use the full 16 bits – a capability Jim designed into the interface hardware but didn't take advantage of in his original driver. After some effort (OK, OK, a *lot* of effort!) they succeeded.

After we had a functional 16-bit driver and prototype board built from the same layout used to have the boards professionally etched and plated, we went ahead and ordered the boards. About six weeks later, Brian Goers received a bouncing baby box of printed circuit boards and we were almost ready to go. Now all we had to do was solder sockets, headers, resistors, and capacitors on the boards and then populate the sockets ... and *then* test these newly constructed boards, all with volunteer GCCC labor.

At the time of finishing this document, we have 50 constructed and fully-tested boards ready to go and waiting only for this documentation. At this point, we've only had a 2% failure rate – that is, we had to build 51 boards to get 50 working boards. We're pretty satisfied with that! The one non-working board had a broken trace – our fault and not the fault of the PCB manufacturer. (whoops)

1.6 Why did it take so long?

First of all, we deeply apologize for running close to two years past our original goal and schedule. Any project that relies on volunteers to get work done is unfortunately subject to delays due to other commitments – especially when you need more than one person at a time to get together. For example, Eddie was finishing his PhD dissertation (which if you would like to see in postscript, just ask him, it's only 4-1/2 megabytes of .ps file) and then starting his new life in the wonderful world of paid employment.

Besides the obvious human-related scheduling problems, we had many (many, too many!) hours wasted trying to track down a hardware problem that didn't exist. (Whoops again) When you are working on a hardware driver, careful attention to the details of the spec is suggested. It helps even more when you can actually *find* a clear definitive spec! Note that the IDE spec has been a moving target since its inception, so while we have tested and proven working the IDE interface with ATA-2 and newer drives, some older drives – those created before the ATA-2 standard was such – may not work with our driver. We encourage anyone to help us correct this deficit by sending us a more inclusive driver; we ran out of time while trying to figure out this problem.

Our final obstacle has been finishing this document, for which we *greatly* appreciate Brett's help. We are not sure if it is fortunate or unfortunate :-) but both Carl and Eddie now have paying jobs; neither had a paying job when this project started. Combining the two schedules, neither, as you might imagine, a "normal" work schedule, has made it difficult for us to meet to finish these last details. For that, we apologize.

Chapter 2

Installing the Interface in your CoCo

2.1 Preparation

First, make sure you have the correct parts! Second, please read this entire chapter before you begin taking anything apart. You should have:

- 1 IDE Interface Card
- This documentation
- 1 LED and LED pigtail
- 1 Jumper used to select interface address
- 1 5.25" disk

You will also need:

- 1 IDE cable, 24" long or shorter – *not longer!*
- 1 IDE hard drive with external case and power supply

You *may* (see below) also need:

- 40-position card edge connector or similar through clever construction

If you wish more information than we have provided here, we will have a Glenside web page devoted to this project in the near future at www.stg.net. Poke around. Once it's there it'll be easy to find. We won't turn away any help you offer.

If your interface does not come with an IDE cable, then make sure that your IDE cable is no longer than 24" (two feet) long. IDE cables are widely and cheaply available. Why do we stress "no longer than 24 inches"? The IDE specification is very strict on this regard; if you are brave you may be lucky. We just won't warrent that you will be lucky!

If you wish to use your IDE interface with a floppy disk controller, without a MultiPak or similar device, you will also need an appropriate card edge connector. If you cannot find a 40-position edge connector, you can construct one from other edge connector parts that are readily available. We describe one possible way to do this later in this chapter.

The floppy disk contains drivers, utilities, and source code for all of the ones written by or modified by Glenside:

- We include an 8-bit driver and a 16-bit driver, each with source. The 8-bit driver is amenable to Nitro and 6309 TFM speedup, but only uses 1/2 of the hard disk capacity. The 16-bit driver can use the entire drive capacity for any drive up to 4 Gig.

- We include, as shareware, the popular Burke&Burke utility "EZGen," along with its documentation. If you use these utilities and do not already own them, we urge you to register your shareware with Chris Burke – do the right thing! You can send shareware donations to Chris Burke at:

Chris Burke
EZGen Shareware Donation
11722 325th Ave
Duvall, WA 98019

If you wish to contact Chris Burke to find out how much he would wish for a donation, you can reach him via EMail at "`serotonin@earthlink.net`"

- We include as freeware (released into the public domain) the following utilities: lformat, recluster, and the source code to both. These utilities are documented below. We also include detect_ide.b09, written and copyrighted by Jim Hathaway and documented below.

2.2 Installing the Interface

Plug the IDE Interface card into your CoCo, Y-Cable, or Multi-Pak. Note that you will need a floppy controller, in addition to the IDE Interface, to finish the installation. However, once your IDE drive and interface are installed and ready, you can do without a floppy controller if you have a special boot ROM allowing you to boot from the hard disk.

Next, connect the IDE hard disk to the controller card using your IDE cable. Be careful to align the cable with pin 1 on the contoller card and pin 1 on the hard drive. Most IDE cables have a red braid on one edge; align that braid with pin 1. If pin 1 is not labeled on your hard drive, pin 1 is practically always the pin closest to the power connector on the hard disk. Pin 1 is labeled on the‘ CoCo IDE Interface card.

Next, connect your LED to the interface card, using the pigtail provided. The LED voltage is turned on and off by the IDE hard disk and *not* by the interface card! If you have a bad hard drive, the LED will not light or may stay on continuously. Note that this is also true of the LED on Intel-based PC systems using Windows and is a feature of IDE! Thus, if you have a bad hard drive, you may initially think the interface card is bad. Please test the interface card with a known-working hard disk before returning the card to us!

The two header pins of the LED connector on the controller card are marked + and -. Although it will not harm your LED to plug it in backwards, it will not light unless you connect it properly. One side of the LED is flat or notched (depending on the LED). Note the orientation marked on the controller card and connect the LED in that orientation.

Finally, make sure you connect your hard drive's power connector!

2.3 Formatting your Disk

You need to know the number of cylinders, heads, and sectors (C/H/S) supported by your hard drive before you can format your hard disk. If this information is not provided printed on the hard drive and you cannot find it from other sources, you can safely make up numbers without any risk. It is important only that $\frac{C*H*2S}{4096}$ is equal to the size of your hard disk in Megabytes. Note that the CoCo stores two 256-byte sectors in one 512-byte sector on the hard disk. This is why you use "2S" in the equation above.

Once you know the "correct" C/H/S values for your hard disk, you are almost ready to format your drive. If your drive is smaller than 128 Meg or if you have a larger drive that you are going to format only to 128 Meg, you can use the standard Microware "format" utility. However, if you wish to format a hard disk to a size larger than 128 Meg, you will need to use a "cluster size" larger than 1 sector per cluster. See the documentation for "lformat" in the next chapter for the details on how to do this.

An option which may be available in the future for large hard disks is partitioning. With partitioning, you divide one physical hard disk into multiple logical hard disks. The current IDE driver does not support partitioning.

2.4 Installing the Driver

The CoCo IDE interface driver exists in four versions: You have the choice of two address ranges for the driver ($FF5x and $FF7x), and you have the choice of using the 8-bit or the 16-bit version of the driver. Almost all people will wish to use the 16-bit version of the driver. The only advantage of the 8-bit driver is that it *could* be (but currently is not) substantially sped up with 6309 instructions. The 16-bit version of the driver cannot be so sped up with 6309 instructions.

2.4.1 Which driver do I want?

If you use a Tandy Multipak, you will want to use the 16-bit $FF5x driver. Otherwise, you will probably want to use the 16-bit $FF7x driver.

2.4.2 Making a boot disk

Make a new boot disk via normal means. We have provided Burke & Burke's "ezgen" as shareware. If you do not currently own ezgen, please do the honorable thing and make the shareware payment to Burke & Burke. Are we sounding annoying yet? :-)

All you need to do to make a new boot disk is to add the hard disk driver and hard disk descriptor to your boot file. Note that if you are currently using a different hard disk adaptor and you wish to use the CoCo IDE Interface *and* another hard disk adaptor, you will need to edit (i.e., "dEd") the "`/h0`" IDE disk descriptor to a name different from you current hard disk descriptors. Note that although it isn't following proper OS-9 rules, the driver current has the base interface address hard coded and does not fetch it from the descriptor. We hope someone will fix this.

Please make a backup of your current working boot disk before you make a new one! Also, while not necessary, you will probably want to make your new boot disk *before* installing the IDE interface.

2.4.3 Testing your new CoCo IDE drive

You can test the CoCo IDE interface without using the driver. To do this, connect the IDE Interface (be sure it is plugged in straight and level!), cable, hard disk, and power to the hard disk. Note that the IDE Interface card *must* be properly supported so that it is level, even for a test run. You do not want the Interface card to slip and short itself out, turning your 6809 or irreplacable 6309 into an expensive 5V fuse.

IDE drives expect to be powered up at the same time as the computer (and interface), not well before. Remember that IDE drives originally were intended only as cheap internal drives for PCs. This means that if you power up your IDE drive well before you power up your CoCo, the drive may not respond to the driver. If this occurs, simply cycle power on your IDE drive and reboot OS-9. (After all, if the drive did not respond to the driver, OS-9 didn't finish booting, did it?) The simplest way to avoid power up problems is to use a power strip for the computer and all external peripherals and drives.

To test your IDE drive without using the driver, just run the "detect_ide.b09" program described in the next chapter. Note that you will need to have BASIC09 to run this program!

To test your IDE drive once you have your new boot disk, run OS-9's "format" command or run the "lformat" command we supply, described in the next chapter. Note that "lformat" has a simple (and simple minded) "physical verify" option.

Once you have the new boot disk and boot up with it, you should be able to format your hard disk. Enjoy!

2.5 Finishing Touches

You will probably wish to put your IDE controller card into a project box of some sort to protect it from dust, nosey neighbors, and errant pets. There are many possibilities available for enclosing your IDE board. We will present a few in this section; they are provided as guides. Your imagination and current CoCo configuration will dictate how you decide to enclose the IDE board in your own system. Wherever possible we will provice JDR, Jameco or Radio Shack part numbers for parts used in this section.

If you are going to have your IDE board plugged into the expansion port of your CoCo, it is important to provide support for the board, to protect it from shorting out, and to protect it from static electricity. The IDE board uses 9 IC's, eight of them are low power CMOS chips which are much more sensitive to static electricity. The primary reason we use these chips is because they consume less power and discharge less heat. The CoCo is known for its wimpy linear power supply and we wanted to ensure that the IDE board would not need a separate power supply.

The CoCo IDE Interface card does not have the "grounding" ears common on many CoCo cartridge packs. Such firm grounding is unnecessary for this card. However, if you are going to hang more devices off the IDE card, you may wish to firmly attach wires directly from the CoCo ground to the ground planes or ground "ears" of the additional devices.

Below, we provide three fairly easy examples of how you might enclose your IDE board. Our personal preference is the first example because it's the easiest to build.

2.5.1 Using the option pads

If you would like to use a floppy disk controller plugged into the CoCo IDE Interface, you will almost certainly want to install a 40-position card-edge connector onto one of the option pads (CONN2 and CONN3) provided on the interface card. We suggest that you use CONN2 for the floppy controller.

40-position PC board card-edge connectors are difficult to find. You may have better luck constructing one. ISA bus (98-position) and PC/XT bus (62-position) card-edge connectors are still readily available. These card-edge connectors have 0.1" contact centers, just as CONN2 and CONN3 on the interface card. Just cut the larger card-edge connectors into smaller pieces, two of which you can combine to create a 40-position connector. You will use the two ends (so as to hold the floppy controller firmly in place, we recommend this!) and will discard part of the middle.

Although these are discontinued parts, you may still possibly find them at Radio Shack stores. The Tandy Part numbers we have available are:

- 98-position: Radio Shack part 276-1454
- 62-position: Radio Shack part 276-1453

We expect that JDR Microdevices and other large venders will also have these parts in stock and readily available. See Appendix E for hardware vendor contact information.

2.5.2 Project Box

PARTS:

- 1 Plastic project box (Radio Shack part 270-1807)
- 4 Sheet metal screws (#8 x 1/2")
- 2 or 4 #6 Stand-offs with a matching number of #6-32 screws and nuts
- 1 piece of foam rubber (optional)
- 4 rubber feet
- 1 LED and LED holder (optional)

TOOLS:

- A drill with the following bits: 1/8", 3/16", 1/4"
- Ruler
- Screw driver(s)
- Fine line marker or pencil

- Saw, nibbler, or file to cut the project box

Step.1 Place your CoCo on a flat surface

Step.2 Put rubber feet on the project box

Step.3 Place project box next to your CoCo as shown below:

Step.4 Mark four spots to screw the project box to your CoCo

Step.5 Drill 3/16" holes in project box

Step.6 Place project box next to your CoCo and mark holes on CoCo

Step.7 Drill 1/8" holes in your CoCo case.

Step.8 Screw the project box to your CoCo

Step.9 Mark where the expansion port slot lines up

Step.10 Remove the project box and cut out the opening for the expansion slot

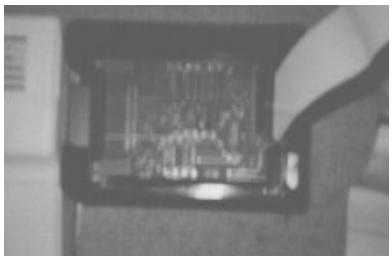
Step.11 Cut a spot at the other end of the project box for the IDE cable

Step.12 Drill several 1/4" holes in the bottom of the project box (if it has feet) for ventilation. Also drill several 1/4" holes in the back of the project box for ventilation.

Step.13 Insert the IDE board into the project box

Step.14 Put the project box back in place next to the CoCo

Step.15 With the IDE board plugged into the CoCo and level (straight out from the expansion slot) measure the height of the board to the bottom of the project box as shown below:



Step.16 Drill two or four 3/32" (or 3/16") holes in the IDE board; choose any locations that do not involve cutting, breaking, shorting, or going through a trace (on *either* side of the board) or chip! Drill matching holes in the bottom of the project box. Make sure that you drill holes that are smaller in diameter than the standoffs that you purchase.

Step.17 Attach the project box to the CoCo with screws. Place foam rubber between the CoCo and the project box, if desired.

Step.18 Insert the IDE board into the CoCo and screw the IDE board into the stand-offs in the project box.

Step.19 Drill a 1/4" hole for the LED wherever you wish to attach the LED. (optional) Use an LED holder to mount the LED.

Step.20 Attach the IDE cable and LED cable to the IDE board.

Step.21 Put the top on the box and attach.

Step.22 Optional - cut slots in the top of the project box for a floppy controller, etc.

Step.23 Enjoy. (Don't forget this step.) For your enjoyment anticipation, here is an example:

2.5.3 Sheet metal box

The second possibility is to make a metal case for the IDE board. This can be done by most shops that do sheet metal work, such as heating/AC shops (they do duct work). If you use a thin enough gauge metal, you can even do it yourself. You will need to use some kind of stand-off to keep the IDE Interface board from shorting itself out by touching the case.

The Disto floppy controller metal cases are an excellent example of what we are suggesting.

2.5.4 Tandy cartridge box

The past possibility we present – we're sure you can think of many more – is to use an old full length Tandy cartridge case. You will still need to do some modifications to make this work; the IDE interface card is longer than a Tandy board. We leave the modifications to your imagination.

Chapter 3

Command Documentation

We provide several useful utilities with source code. Each is describe below. The "lformat" and "recluster" programs were written by Eddie Kuns. These programs are released into the public domain. Jim Hathaway wrote the Basic09 program "detect_ide.b09" below and this program is copyrighted, 1996 by Jim.

3.1 lformat

The usage output for "lformat" is listed below:

```
Usage: lformat [/device] [c/h/s] -spc "drive name" [-v]
c/h/s = cylinders/heads/sectors-per-track
        e.g., 615/2/34 (slashes necessary)
        sectors-per-track is twice the PC value
        (256 byte sectors, not 512 byte sectors)
spc =   sectors per cluster (dash necessary)
quotes necessary for drive name
Defaults:
       device:     /h0
       spc:        2
       c/h/s:      read from device descriptor
       drive name: "CoCo IDE Drive"
Sectors per cluster MUST be a power of 2 (1 up to 128)
Options -h and -? produce this help screen
Option -v indicates do physical verify
Option -r indicates 'ready' -- automatic yes to all prompts
```

You can use "lformat" to format a hard disk with any cluster size. The default cluster size (spc = sectors per cluster) for lformat is two sectors per cluster. If you try to format a hard disk with a cluster size that is too small to format the entire drive, the program will exit with an informative error message.

Note that lformat will *not* perform a hard (low level) format on an IDE drive. Thus, you cannot use lformat to change the sector size on an IDE drive. The driver for the CoCo IDE controller assumes the drive is formatted for 512 byte sectors; the driver translates these 512-byte sectors into OS-9/6809 256 byte sectors. The largest hard disk that OS-9 can format with one sector per cluster is 128 Meg. For the curious, this limit exists because OS-9 allows only 64 kbytes for the cluster allocation bitmap: 64 kbytes times eight bits per byte times 256 bytes per minimum disk allocation unit equals 128 Meg. With two sectors per cluster, you can format a 256 Meg drive, and so on.

OS-9 allows the cluster size to be any power of two between 1 and 128 sectors per cluster; however, OS-9 cannot handle any disk drive larger than 2 Gig unless the drive is partitioned into multiple "logical" drives. Technically, the limit is 4 Gig, but certain OS-9 tools (such as dcheck) use relative seeks on the raw disk device (/h0@) and a seek cannot seek past 2 Gig. The 4-byte integer passed to "seek" is a *signed* integer.

Of course, "lformat" will erase any information that is previously on the drive, just as the normal OS-9 "format" program does. The difference is that the standard OS-9/6809 "format" command will not format a disk with a cluster size other than 1.

3.2 recluster

The usage output for "recluster" is shown below:

```
Usage: recluster [/device] [new sectors/cluster]
       default device is /h0
       default number of sectors per cluster is 2
Must specify device or sectors/cluster or both
Sectors per cluster MUST be a power of 2 (up to 128)
```

Recluster is not as useful as lformat, but it is included for completeness. Recluster will change the cluster size on a blank and freshly formatted disk. Recluster does not format the disk itself, so is not useful for disks or partitions larger than 128 Meg.

3.3 detect_ide.b09

This basic09 program will talk to an IDE hard drive without using the hard disk driver. It is an easy way to find out how many heads, cylinders, and sectors per track a given hard drive has, if you don't have that information already available. The version of the program that is provided only searches for the first hard drive and is hard coded to use the $FF7x address range. If you wish to use this program in the $FF5x address range, you will need to change the PEEK and POKE statements, subtracting $20 (or 32, since the addresses in the program are in decimal, not hex, notation) from each address.

Chapter 4

How it works:

4.1 Communicating with the drive

The physical interface consists of a 3 (or 4) pin power connector and a 40-pin ribbon cable which will control up to two drives connected in a 'daisy chain' fashion. A complete listing of ribbon cable signals sorted into functional groups, along with a short description of each group, may be found in Appendix A. For our purposes here we need only note that these signals provide a means for the host software to communicate with the drive(s) through two banks of virtual registers, the Control Block Registers and the Command Block Registers, located on the drive(s).

In fact, an IDE controller is a glorified "data buffer" and not a controller – one can communicate directly with a controller (e.g., a SCSI controller) whether a drive is connected to it or not, but this is not the case for an IDE controller. Remember that IDE stands for Intelligent Drive Electronics, meaning that the intelligence is put onto the drive's embedded controller.

4.1.1 The Command Block Registers

The Command Block Registers are used for sending commands to the drive or posting status from the drive.

The Command Block Registers are the primary means of communication between the host software and the drive(s) electronics and hence form the heart of the IDE. For purposes of explanation the Command Block Registers may be roughly divided into three (somewhat arbitrary) groups; Interface control, Device control, and Sector address. Below is a brief description of each of these groups, a table of addresses for the registers in that group, and a short synopsis of each registers function. Further details may be found in Appendix C.

4.1.1.1 The Interface Control group

- consists of the read-only Status, write-only Command, and read-write Device/Head registers.

Interface Control

CS1FX-	CS3FX-	DA2	DA1	DA2	READ (DIOR-)	WRITE (DIOW-)
A	N	1	1	0	Drive/Head	Drive/Head
A	N	1	1	1	Status	Command
A	A	x	x	x	Invalid Address	Invalid Address

Commands are issued by writing a code to the Command register. The Status register is used report the drive state (ie. drive busy, command complete, command error, etc.). The High order bits of the Drive/Head register select the drive and addressing mode, and must be set before any other operations make sense, the only exception to this is the Execute Drive Diagnostic command. which is described below. The 4 low order bits are head select bits and will be described with the Sector Address group of registers below. Once the

drive and addressing mode have been selected, the host may issue a command by writing to the command register.

4.1.1.2 The Device Control group

- includes the read-only Error register and the write-only Features register, as well as the read-write Data and Sector Count registers.

Device Control

CS1FX-	CS3FX-	DA2	DA1	DA0	READ (DIOR-)	WRITE (DIOW-)
A	N	0	0	0	Data	Data
A	N	0	0	1	Error register	Features
A	N	0	1	0	Sector count	Sector count

The Data register is used to move information to and from the drive. The features register provides a means to access special features which some drive vendors provide. The Error register reports details on the type of error when one occurs. The Sector Count register specifies how many sectors are to be transferred on a read or write command.

4.1.1.3 The Sector Address registers

- specify which sector(s) on the selected surface are to be transferred. Since the names and interpretation of these registers depends on which of the two available addressing modes are used, they will be described as a set.

Sector Address

CS1FX-	CS3FX-	DA2	DA1	DA0	CHS mode	LBA mode
A	N	0	1	1	Sector number	LBA bits 0-7
A	N	1	0	0	Cylinder low	LBA bits 8-15
A	N	1	0	1	Cylinder high	LBA bits 16-23
A	N	1	1	0	Drive/Head	LBA bits 24-27

The IDE specification provides for two different methods of addressing sectors, the traditional Cylinder/Head/Sector (C/H/S) method and a more general "Logical Block Address" method. In LBA mode, the concatenated Sector Address registers are treated as a 28-bit binary address for the desired sector, in other words 0000040h would specify the "sixty-fourth (40 hex)" sequential sector on the drive, regardless of which cylinder or head it happens to fall on. The current CoCo IDE driver uses the C/H/S method with translation as described in section 3.3.

4.1.2 The Control Block Registers

The Control Block Registers are used for drive control and to post alternate status. Since they are only really useful for drives operating in Interrupt driven I/O mode (which the Glenside board doesn't support) they are described in Appendix B which lists these registers, the addresses that select them, and uses them as an example to show how the interface signals address and access the Drive registers.

4.2 Interface Initialization

After reset or power-on Drive 0 sets the BSY bit in it's status register, performs it's hardware initialization, and waits for Drive 1 to report. It then clears the BSY bit and waits for the host.

The very first thing the host has to know is how many working IDE drives are connected to the interface. This is not so straightforward a query as might be supposed. As implied above, the IDE interface is incredibly

stupid. It is capable of supplying the information needed to answer this simple query, but only if the host can issue the correct command (Execute Drive Diagnostic), read the appropriate registers (which may include the status and error registers of one or both drives) and then interpret and correlate the relevant bits to arrive at one of the following conclusions;

- No drives present
- Drive 0 present, not working.
- Drive 0 present, working
- Both drives present, Drive 0 working
- Both drives present, Drive 1 working
- Both drives present, both working

The host does this by issuing the "Execute Drive Diagnostic" command (ie by writing 90h to the command register). The "Execute Drive Diagnostic" command is unique in several respects. For all other commands both drives monitor the command register but only the currently selected drive executes commands. The "Diagnostic" command is executed by both drives and hence can be issued before selecting a drive (although the host does have to select a drive before it can read the results of the command from the status and error registers). Also, at the end of this command, the error register bits are redefined to contain a diagnostic code which indicates what type of internal drive failure (if any) occurred.

After issuing the command the host waits up to 6 seconds for Drive 0 to clear the BSY bit in it's status register. The host can then read the Drive 0 error register to get the results of the diagnostics. If the high bit is set it means Drive 1 is present but did not pass it's diagnostics. The remaining bits provide the Drive 0 diagnostic code. The Drive 1 diagnostic code (if relevant) may be read from the Drive 1 error register. If Drive 1 is not present then Drive 0 presents a dummy Drive 1 status register with all bits clear.

Some time after clearing the BSY bit (thus allowing the host to access the command block registers) Drive 0 will set the DRDY bit to indicate it is ready to execute another command. The ata-3 standard recommends the host allow at least 30 seconds for this.

4.3 Drive Initialization

Now that the host software knows it has at least one drive to talk to the next step is to determine what kind(s) of drive it is dealing with (ie. Drive size and geometry, what type(s) of I/O it can support, which special features or options are available, etc....). In the early days of IDE this was accomplished by setting a "Type" parameter in the BIOS when the drive was installed. On boot the BIOS would get the Type number and look in it's table of known drive types to get the size and geometry which it reported to the OS. With the explosive growth in capacity of IDE drives, as well as the addition of a welter of optional features and higher speed I/O options this method of selecting from a limited number of predefined drive types proved unsatisfactory. This problem was addressed in the ATA-2 specification by adding the "Identify Device," "Initialize Device Parameters," and "Set Features" commands.

The Initialize Device Parameters command tells the drive to emulate some specified geometry regardless of it's own physical layout (The geometry of a drive refers to how many heads, how many tracks (cylinders), and how many sectors per track it has). Remapping in this way allows the host to access drives that would otherwise be incompatible (e.g.. if there are more physical cylinders than the host can address, remapping makes the drive appear to have fewer cylinders but more heads and/or more sectors per track). Using this feature also simplifies (greatly!) the design of the CoCo IDE driver. (Not to mention allowing faster access because the driver does not have to do C/H/S conversions for each access.) For details on the use of the Initialize Device Parameters command see the standard.

The Set Features command is used to enable, disable, or set the parameters for any of the many optional features supported by various drive manufacturers. The optional features available on a given drive are reported by the Identify Device command below. A description of all the options supported by the Set

Features command would be far beyond the scope of this document so I again refer the interested reader to the standard.

When the drive receives an Identify Device command it fills its sector buffer with a table of drive parameters and sends it to the host, this is where it reports it's geometry, as well as a welter of details about it's performance, operation, optional features, I/O modes supported, cache type and size, etc. The meaning of a given entry in this table is determined by it's location in the sector buffer, so for example the number of user addressable logical cylinders in the default translation mode could be found by looking at word 1 (that is bytes 3 and 4) of the table. Once again, for full details on the information supplied by the Identify Device command and the layout of the table it returns you are referred to the standard.

4.4 Drive Operation

Now that the host has determined which drives are available, and made sure that it and the drive(s) are on the same page as far as I/O mode and logical drive geometry are concerned, it can finally access the drive(s). Before describing drive access however there are a few things it is useful to know. This section describes how an IBM PC AT compatible computer (running any operating system: DOS, Windows, OS-9000, Solaris, or Linux!) boots from a hard drive. AT computers boot from a BIOS in ROM which specifically follows the steps described below. Your milage may vary; note that OS-9 on a CoCo does not follow these specific steps! This information is presented as background.

4.4.1 Booting

Although it is not technically a part of the standard, the first sector on a drive is conventionally given a special status and called the Master Boot Record (MBR). Up till now all of the host commands have been issued by a low-level bootstrap program (which may reside in ROM, or have been loaded from some other device, or even entered by hand). The MBR contains the information necessary for this low-level boot program to begin the process of booting the Operating System. Specifically, it contains the Partition Table and a Boot Loader.

4.4.1.1 The Partition Table

Partitions are a way of splitting one physical drive into several distinct logical drives. There may be any number of reasons for doing this, for example you want to share one drive between two different Operating Systems, or maybe you have a 2-Gig drive and your Operating System can't handle drives any larger than 500-Meg, or perhaps you want (need) to set aside part of your drive as virtual memory so you can run larger applications. In any case the Partition Table allows you to do this, it lists the location, size, and type of each partition (amongst other things). There are many details of Partitions and the Partition Table that I have glossed over or ignored here, but for our purposes the essential points are these.

1. Each partition looks like a separate drive to the OS.
2. It is possible to have different OS's on different Partitions.

4.4.1.2 The Boot Loader

The boot loader is a small program (Both it and the Partition Table must fit into one 512-byte sector) who's purpose in life is to bridge the gap between the low-level (often BIOS ROM based) boot and the more flexible Disk-based boot used by most Operating Systems. Since it lives on the MBR, which is always the first physical sector of the drive, the low-level boot always knows where to load it from. Since it shares it's home with the Partition Table, it always knows where each partition begins. The first sector of each bootable Partition contains yet another bootstrap program which is responsible for starting the OS that resides on that particular partition. Thus the Boot Loader is responsible for loading the first sector of whichever partition is to be booted and executing it. This two stage process is what allows different OS's to share a physical drive, since each OS has it's own bootstrap program nestled safely at the beginning of it's own partition. Some boot loaders always boot the first partition marked as bootable, others allow the operator to specify which partition to boot from.

4.4.2 Reading the MBR

As just explained, the first thing we need to do after initializing all the hardware is to read the MBR which is always located at Sector 1 of Cylinder 0 on Head 0. We'll also assume we're booting from Drive 0 and using the traditional C/H/S address mode.

4.4.2.1 Selecting the Drive

Before we load any other parameters we need to tell the interface which drive we want to use, however, a quick look at the bit definitions for the Drive/Head register reveals a problem.

Drive/Head Register bit definitions

b7	b6	b5	b4	b3	b2	b1	b0
1	L	1	DRV	HS3	HS2	HS1	HS0

The Drive, Head, and Addressing Mode select bits share a register (see Appendix C for full details). We need to combine these three parameters into one byte which will then be loaded into the Drive/Head register. According to our table bits 7 and 5 should always be 1, bit 6 selects between C/H/S and LBA addressing mode, bit 4 selects between Drive0 and Drive1, and bits 0-3 select the head. Fortunately for us (actually by design), the only bits that change after the drive is determined are the Head Select bits, so we can define a mask for each drive which has the appropriate bits set and then do a logical 'or' of this mask with the head number to generate the necessary register value. In our example we use a mask of C0 hex (10100000 binary; C/H/S mode, Drive 0) and 0 hex (binary 0000) for our head number. 'Or'ing these two together gives us C0 hex as our register value (if we had wanted head number 8, we would have ended up with C8 hex as a register value) which we load into the register (Appendix B gives a description of reading and writing the Control Block Registers, accessing the Command Block Registers is similar).

4.4.2.2 Setting the other Parameters

Setting the cylinder and sector is fairly straightforward, the only complication is that the 16-bit cylinder address has to be split into two 8-bit bytes before being loaded into the appropriate registers (of course for the MBR both parts of the cylinder address are 0). It is also important to remember that unlike the Cylinder, Head, and Drive addresses (which start with 0) the sector addresses start with 1.

4.4.2.3 Issuing the command

Reading the MBR is not only the first command we need to use, it is also the simplest. We only want to read one sector, so we probably don't have to worry about the Sector Count parameter (although to be safe we should set it to 01 hex). All that remains is to load the command register with either 20 hex (PIO mode Read Sector with retries) or 21 hex (without retries) and call the loop that monitors the status register and reads a byte (word) from the data register each time DRQ is asserted.

4.4.3 Other Commands

Appendix D contains two tables, one is a list of Mandatory Commands for the ATA-1 and ATA-3 specs, the other is a list of the ATA-3 commands that gives their protocol, type, and command code. Note that commands listed as mandatory do not have to be implemented by the host. Because of the hardware limitations of the CoCo it was not practical to provide Interrupt driven I/O nor was it feasible to implement the DMA mode data transfers. The use of the Write Sector(s) commands is similar to that described for Read Sector(s) above. The Execute Drive Diagnostics command was described above under Initialization. Details of usage for the remaining commands is going to vary according your particular OS and Drive combination, and this would not only take us beyond the scope of this document, it is also much farther outside my area of expertise than I am willing to speculate (these two boundaries are not unrelated).

4.5 Final Note

When Carl first asked me to write this portion of the Glenside IDE documentation he asked for something like "A brief layman's description of the IDE Registers". It doesn't seem to be particularly brief, although I have totally ignored or glossed over innumerable complicating details. It is also highly debatable how accessible this will be to those without at least some prior knowledge of disk-computer interfaces. Nevertheless it has been my intention to provide enough background to give novices a place to start, enough scope to allow the more experienced to see how the IDE fits in with the rest of their system, and enough detail (mostly in the Appendices) to make this a useful reference for experts. If I have succeeded in any one of these goals then I will be satisfied, more than that I can't reasonably ask.

Appendix A

Interface signals description

NOTE: A trailing dash on a signal name indicates it is active low.

A.0.0.1 Register Block Select

The drive registers are accessed in two banks, the Command Block registers and the Control Block registers, CS1FX- selects the Command Block registers, CS3FX- selects the Control Block registers.

Signal Name	Pin#	Direction	Description
CS1FX-	37	Host–>Drive	Drive chip select 0
CS3FX-	38	Host–>Drive	Drive chip select 1

A.0.0.2 Drive Register Address

Drive address 0-2 are used to address a particular read/write register pair within the selected bank.

Signal Name	Pin #	Direction	Description
DA0	35	Host–>Drive	Drive address bus - bit 0
DA1	33	Host–>Drive	Drive address bus - bit 1
DA2	36	Host–>Drive	Drive address bus - bit 2

A.0.0.3 Drive Data Bus

DD0-15 form a Data bus to communicate with drive registers.

NOTE: 8 bit transfers use the low order bits of the bus.

Signal Name	Pin #	Direction	Description
DD0	17	Host<->Drive	Drive data bus - bit 0
DD1	15	Host<->Drive	Drive data bus - bit 1
DD2	13	Host<->Drive	Drive data bus - bit 2
DD3	11	Host<->Drive	Drive data bus - bit 3
DD4	9	Host<->Drive	Drive data bus - bit 4
DD5	7	Host<->Drive	Drive data bus - bit 5
DD6	5	Host–>Drive	Drive data bus - bit 6
DD7	3	Host–>Drive	Drive data bus - bit 7
DD8	4	Host–>Drive	Drive data bus - bit 8
DD9	6	Host–>Drive	Drive data bus - bit 9
DD10	8	Host–>Drive	Drive data bus - bit 10
DD11	10	Host–>Drive	Drive data bus - bit 11
DD12	12	Host–>Drive	Drive data bus - bit 12
DD13	14	Host–>Drive	Drive data bus - bit 13
DD14	16	Host–>Drive	Drive data bus - bit 14
DD15	18	Host–>Drive	Drive data bus - bit 15

A.0.0.4 I/O Control Signals

These signals control the timing and direction of data flowing between the computer and the selected drive (many of them deal with capabilities that are not implemented for the Glenside IDE)

Signal Name	Pin #	Direction	Description
DIOR-	25	Host–>Drive	Drive I/O read
DIOW-	23	Host–>Drive	Drive I/O write
DMACK-	29	Host–>Drive	DMA acknowledge
DMARQ	21	Host<–Drive	DMA request
INTRQ	31	Host<–Drive	Drive interrupt
IOCS16-	32	Host<–Drive	Drive 16-bit I/O
IORDY	27	Host<–Drive	I/O channel ready

A.0.0.5 Reset

Hardware Reset, force drive(s) to reinitialize.

Signal Name	Pin #	Direction	Description
RESET-	1	Host–>Drive	Drive reset

A.0.0.6 Initialization

These two signals are time multiplexed between the two drives and are used to report drive status during initialization

Signal Name	Pin #	Direction	Description
DASP-	39	Host<->Drive	Drive active/Drive 1 present
PDIAG-	34	Host<->Drive	Passed diagnostics

A.0.0.7 Connector keying

Pin 20 on the cable connector may be plugged to insure correct orientation.

Signal Name	Pin #	Direction	Description
keypin	20	N/A	Pin used for keying the interface connector

A.0.0.8 Optional

SPSYNC and CSEL implement optional features, see the standard for a description.

Signal Name	Pin #	Direction	Description
SPSYNC:	28	N/A	Spindle sync
CSEL	28	N/A	Cable select

Appendix B

Control Block Registers

B.1 Register Access:

From the viewpoint of the host software the interface appears as two banks of virtual registers, the Command Block Registers and the Control Block Registers. The physical specifications provide seven signals (in addition to the data bus) to control register access, they are;

DA0-DA2 This is the binary encoded Drive Address bus. It is used to specify which of the eight possible read/write register pairs within the currently enabled block is to be accessed.

CS1FX= Holding this signal low enables the Command Block Registers for access.

CS3FX- Holding this signal low enables the Control Block Registers for access.

DIOR- This is the read strobe, the host brings it low to begin a read cycle. The information on the data bus is latched by the rising edge of this signal which ends the cycle.

DIOW- This is the write strobe, it operates similarly to DIOR-, bringing it low begins a write cycle, bringing it back hi ends the cycle and causes the drive to latch the contents of the data bus into the currently addressed register.

B.2 The Control Block Registers

As of (draft) ata3r7b there are only two Control Block Registers defined. Their addresses are listed in the table below.

Logic conventions are:

A = signal asserted;

N = signal negated;

x = does not matter which it is.

CS1FX	CS3FX	DA2	DA1	DA0	READ (DIOR-)	WRITE (DIOW-)
N	N	x	x	x	Data bus high imped	Not used
N	A	0	x	x	Data bus high imped	Not used
N	A	1	0	x	Data bus high imped	Not used
N	A	1	1	0	Alternate status	Device control
N	A	1	1	1	Drive Address (obsolete)	Not used

B.2.1 Alternate status register

The Alternate Status Register is read by setting DA2-DA0 to 110, CS3FX- to 0 and initiating a read cycle with DIOR-. The Device Control Register is accessed by initiating a write cycle (at the same address as the alternate status register) with DIOW-. These two registers provide for what might be called 'auxiliary' control of the interface.

The bits in the read-only alternate status register are identical to the bits in the status register (in the Command Block) and will be described there, the only difference between the status and alternate status registers is that reading the alternate status does not acknowledge an interrupt nor clear pending interrupts as does a read of the status register.

b7	b6	b5	b4	b3	b2	b1	b0
BSY	DRDY	DWF	DSC	DRQ	CORR	IDX	ERR

See 7.2.13 for definitions of the bits in this register.

B.2.2 Device control register

The other Control Block register is the write-only device control register. Of the 8 bits in the device control register only 3 matter, the upper 5 bits (7-3) are reserved. Bit 2 (SRST) is the software reset bit, as long as it is 1 the drive(s) are held reset. Bit 1 (nIEN) is the interrupt enable bit, when nIEN is 0 the selected drive is allowed to make an interrupt request to the host by raising conductor 31 (INTRQ) of the ribbon cable to a logical 1, when nIEN is 1 interrupts are not allowed. The standard specifies that Bit 0 must always written with a 0.

The bits in this register are as follows:

b7	b6	b5	b4	b3	b2	b1	b0
x	x	x	x	1	SRST	nIEN	0

- SRST is the host software reset bit. The drive is held reset when this bit is set. If two disk drives are daisy chained on the interface, this bit resets both simultaneously. Drive 1 is not required to execute the DASP-handshake procedure.
- nIEN is the enable bit for the drive interrupt to the host. When nIEN=0, and the drive is selected, INTRQ shall be enabled through a tri-state buffer. When nIEN=1, or the drive is not selected

B.2.3 Drive address register

As of ATA3R7b this register is considered obsolete, it is included here for completeness.

This register contains the inverted drive select and head select addresses of the currently selected drive. The bits in this register are as follows:

b7	b6	b5	b4	b3	b2	b1	b0
HiZ	nWTG	nHS3	nHS2	nHS1	nHS0	nDS1	nDS0

- HiZ shall always be in a high impedance state.
- nWTG is the Write Gate bit. When writing to the disk drive is in progress, nWTG=0.
- nHS3 through nHS0 are the one's complement of the binary coded address of the currently selected head. For example, if nHS3 through nHS0 are 1100b, respectively, head 3 is selected. nHS3 is the most significant bit.
- nDS1 is the drive select bit for drive 1. When drive 1 is selected and active, nDS1=0.
- nDS0 is the drive select bit for drive 0. When drive 0 is selected and active, nDS0=0.

NOTE 5 - Care should be used when interpreting these bits, as they do not always represent the expected status of drive operations at the instant the status was put into this register. This is because of the use of caching, translate mode and the Drive 0/Drive 1 concept with each drive having its own embedded controller.

Appendix C

Command Block Register Details

C.1 Interface Control Registers

C.1.1 Command register

This register contains the command code being sent to the drive. Command execution begins immediately after this register is written. The executable commands, the command codes, and the type and I/O mode of each command are listed in Appendix D.

b7	b6	b5	b4	b3	b2	b1	b0

C.1.2 Drive/head register

This register contains the drive and head numbers. The contents of this register define the number of heads minus 1, when executing an Initialize Drive Parameters command.

b7	b6	b5	b4	b3	b2	b1	b0
1	L	1	DRV	HS3	HS2	HS1	HS0

- L is the binary encoded address mode select. When L=0, addressing is by CHS mode.When L=1, addressing is by LBA mode.
- DRV is the binary encoded drive select number. When DRV=0, Drive 0 is selected. When DRV=1, Drive 1 is selected.
- If L=0, HS3 through HS0 contain the binary coded address of the head to be selected e.g., if HS3 through HS0 are 0011b, respectively, head 3 will be selected. HS3 is the most significant bit. At command completion, these bits are updated to reflect the currently selected head.
- If L=1, HS3 through HS0 contain bits 24-27 of the LBA. At command completion, these bits are updated to reflect the current LBA bits 24-27

C.1.3 Status register

This register contains the drive status. The contents of this register are updated at the completion of each command. When BSY is cleared, the other bits in this register shall be valid within 400 nsec. If BSY=1, no other bits in this register are valid. If the host reads this register when an interrupt is pending, it is considered to be the interrupt acknowledge. Any pending interrupt is cleared whenever this register is read.

NOTE 6 - If Drive 1 is not detected as being present, Drive 0 clears the Drive 1 Status Register to 00h (indicating that the drive is Not Ready).

b7	b6	b5	b4	b3	b2	b1	b0
BSY	DRDY	DWF	DSC	DRQ	CORR	IDX	ERR

NOTE 7 - Prior to the definition of this standard, DRDY and DSC were unlatched real time signals.

- BSY (Busy) is set whenever the drive has access to the Command Block Registers. The host should not access the Command Block Register when BSY=1. When BSY=1, a read of any Command Block Register shall return the contents of the Status Register. This bit is set by the drive (which may be able to respond at times when the media cannot be accessed) under the following circumstances:

a) within 400 nsec after the negation of RESET- or after SRST has been set in the Device Control Register. Following acceptance of a reset it is recommended that BSY be set for no longer than 30 seconds by Drive 1 and no longer than 31 seconds by Drive 0.

b) within 400 nsec of a host write of the Command Register with a Read, Read Long, Read Buffer, Seek, Recalibrate, Initialize Drive Parameters, Read Verify, Identify Drive, or Execute Drive Diagnostic command.

c) within 5 μsecs following transfer of 512 bytes of data during execution of a Write, Format Track, or Write Buffer command, or 512 bytes of data and the appropriate number of ECC bytes during the execution of a Write Long command.

- DRDY (Drive Ready) indicates that the drive is capable of responding to a command. When there is an error, this bit is not changed until the Status Register is read by the host, at which time the bit again indicates the current readiness of the drive. This bit shall be cleared at power on and remain cleared until the drive is ready to accept a command.

- DWF (Drive Write Fault) indicates the current write fault status. When an error occurs, this bit shall not be changed until the Status Register is read by the host, at which time the bit again indicates the current write fault status.

- DSC (Drive Seek Complete) indicates that the drive heads are settled over a track. When an error occurs, this bit shall not be changed until the Status Register is read by the host, at which time the bit again indicates the current Seek Complete status.

- DRQ (Data Request) indicates that the drive is ready to transfer a word or byte of data between the host and the drive.

- CORR (Corrected Data) indicates that a correctable data error was encountered and the data has been corrected. This condition does not terminate a data transfer.

- IDX (Index) is set once per disk revolution.

- ERR (Error) indicates that an error occurred during execution of the previous command. The bits in the Error Register have additional information regarding the cause of the error.

C.2 Device Control Registers

C.2.1 Data register

This 16-bit register is used to transfer data blocks between the device data buffer and the host. It is also the register through which sector information is transferred on a Format Track command. Data transfers may be either PIO or DMA.

C.2.2 Error register

This register contains status from the last command executed by the drive or a Diagnostic Code.

At the completion of any command except Execute Drive Diagnostic, the contents of this register are valid when ERR=1 in the Status Register. Following a power on, a reset, or completion of an Execute Drive Diagnostic command, this register contains a Diagnostic Code (see standard for Diagnostic Codes).

b7	b6	b5	b4	b3	b2	b1	b0
BBK	UNC	MC	IDNF	MCR	ABRT	TK0NF	AMNF

- BBK (Bad Block Detected) indicates a bad block mark was detected in the requested sector's ID field.
- UNC (Uncorrectable Data Error) indicates an uncorrectable data error has been encountered.
- MC (Media Changed) indicates that the removable media has been changed i.e., there has been a change in the ability to access the media.
- IDNF (ID Not Found) indicates the requested sector's ID field could not be found.
- ABRT (Aborted Command) indicates the requested command has been aborted due to a drive status error (Not Ready, Write Fault, etc.) or because the command code is invalid.
- MCR (Media Change Requested) indicates that the release latch on a removable media drive has been pressed. This means that the user wishes to remove the media and requires an action of some kind e.g., have software issue a Media Eject or Door Unlock command.
- TK0NF (Track 0 Not Found) indicates track 0 has not been found during a Recalibrate command.
- AMNF (Address Mark Not Found) indicates the data address mark has not been found after finding the correct ID field.

C.2.3 Features register

This register is command specific and may be used to enable and disable features of the interface e.g., by the Set Features Command to enable and disable caching. This register may be ignored by some drives. Some hosts, based on definitions prior to the completion of this standard, set values in this register to designate a recommended Write Precompensation Cylinder value.

C.2.4 Sector count register

This register contains the number of sectors of data requested to be transferred on a read or write operation between the host and the drive. If the value in this register is zero, a count of 256 sectors is specified. If this register is zero at command completion, the command was successful. If not successfully completed, the register contains the number of sectors which need to be transferred in order to complete the request. The contents of this register may be defined otherwise on some commands e.g., Initialize Drive Parameters, Format Track or Write Same commands.

C.3 Sector Address Registers

C.3.1 Drive/Head register

See C.1.2 above.

C.3.2 Cylinder high register

This register contains the high order bits of the starting cylinder address for any disk access. At the end of the command, this register is updated to reflect the current cylinder number. The most significant bits of the cylinder address shall be loaded into the cylinder high Register.

In LBA Mode this register contains Bits 16-23. At the end of the command, this register is updated to reflect the current LBA Bits 16-23.

NOTE 4 - Prior to the introduction of this standard, only the lower 2 bits of this register were valid, limiting cylinder address to 10 bits i.e., 1024 cylinders.

C.3.3 Cylinder low register

This register contains the low order 8 bits of the starting cylinder address for any disk access. At the end of the command, this register is updated to reflect the current cylinder number.

In LBA Mode this register contains Bits 8-15. At the end of the command, this register is updated to reflect the current LBA Bits 8-15.

C.3.4 Sector number register

This register contains the starting sector number for any disk data access for the subsequent command. The sector number may be from 1 to the maximum number of sectors per track.

In LBA Mode this register contains Bits 0-7. At the end of the command, this register is updated to reflect the current LBA Bits 0-7.

See the command descriptions (in the standard) for contents of the register at command completion (whether successful or unsuccessful).

Appendix D

IDE Commands

D.1 ATA Mandatory Commands

These are commands that an IDE standard device is required to implement, this does not mean that the host software must utilize all of these functions.

ATA-1	ATA-3
Execute drive diagnostic	Execute Device Diagnostic
Format track	Identify Device
Initialize drive parameters	Initialize Device Parameters
Read long (w/retry)	Read DMA (w/ retry)
Read long (w/o retry)	Read DMA (w/o retry)
	Read Multiple
Read sector(s) (w/retry)	Read Sector(s) (w/ retry)
Read sector(s) (w/o retry	Read Sector(s) (w/o retry)
Read verify sector(s) (w/retry)	Read Verify Sector(s) (w/ retry)
Read verify sector(s) (w/o retry)	Read Verify Sector(s) (w/o retry)
Recalibrate	Set Features
Seek	Set Multiple Mode
Write long (w/retry)	Write DMA (w/ retry)
Write long (w/o retry)	Write DMA (w/o retry)
	Write Multiple
Write sector(s) (w/retry)	Write Sectors (w/ retry)
Write sector(s) (w/o retry)	Write Sectors (w/o retry)

D.2 ATA-3 Commands

Command	Protocol	Type	Command Code
Check Power Mode	ND	O	98h or E5h
Door Lock	ND	O	DEh
Door Unlock	ND	O	DFh
Download Microcode	PO	O	92h
Execute Device Diagnostic	ND	M	90h
Format Track	VS	V	50h
Identify Device	PI	M	ECh
Identify Device DMA	DM	O	EEh
Idle	ND	O	97h or E3h
Idle Immediate	ND	O	95h or E1h
Initialize Device Parameters	ND	M	91h
Media Eject	ND	O	EDh
Nop	ND	O	00h
Read Buffer	PI	O	E4h
Read DMA (w/ retry)	DM	M	C8h
Read DMA (w/o retry)	DM	M	C9h
Read Long (w/ retry)	PI	O	22h
Read Long (w/o retry)	PI	O	23h
Read Multiple	PI	M	C4h
Read Sectors(s) (w/ retry)	PI	M	20h
Read Sectors(s) (w/o retry)	PI	M	21h
Read Verify Sector(s) (w/ retry)	ND	M	40h
Read Verify Sector(s) (w/o retry)	ND	M	41h
Recalibrate	ND	O	10h
Security Disable Password	PO	O	F6h
Security Erase Prepare	ND	O	F3h
Security Erase Unit	PO	O	F4h
Security Freeze	ND	O	F5h
Security Set Password	PO	O	F1h
Security Unlock	PO	O	F2h
Seek	ND	M	70h
Set Features	ND	M	EFh
Set Multiple Mode	ND	M	C6h
Sleep	ND	O	99h or E6h
Smart Disable Operations	ND	O	B0h*
Smart Enable/Disable Autosave	ND	O	B0h*
Smart Enable Operations	ND	O	B0h*
Smart Read Thresholds	PI	O	B0h*
Smart Read Values	PI	O	B0h*
Smart Return Status	ND	O	B0h*
Smart Save Values	ND	O	B0h*
Standby	ND	O	96h or E2h
Standby Immediate	ND	O	94h or E0h
Write Buffer	PO	O	E8h
Write DMA (w/ retry)	DM	M	CAh
Write DMA (w/o retry)	DM	M	CBh
Write Long (w/ retry)	PO	O	32h
Write Long (w/o retry)	PO	O	33h
Write Multiple	PO	M	C5h
Write Sector(s) (w/ retry)	PO	M	30h
Write Sector(s) (w/o retry)	PO	M	31h
Write Verify	PO	O	3Ch
Others (vendor specific)	VS	V	

* The “Smart” commands pass other parameters in the features and cylinder registers.

Protocols:

- ND - Non Data
- DM - DMA
- VS - Vendor Specific
- PI - PIO data input
- PO - PIO data output

Types:

- M - Mandatory
- O - Optional
- V - Vendor Specific

Appendix E

Hardware Details

E.1 Hardware Vendor Contact Information

For your convenience, we provide the following contact information. We do not necessarily endorse any specific vendor! However, we used these vendors to obtain the parts for this project.

Jameco Electronic Components
1-800-831-4242
`http://www.jameco.com`
info@jameco.com

JDR Microdevices
1-800-538-5000
`http://www.jdr.com`

E.2 Parts List

The IDE Interface uses the following chips:

- U1 - 74HCT245
- U2 - 74HCT573
- U3 - 74HCT573
- U4 - 74HCT32
- U6 - 74HCT04
- U7 - 74HCT00
- U8 - 74HCT138
- U9 - 74LS133
- U10 - 74HCT574

The IDE Interface uses the following capacitor value:

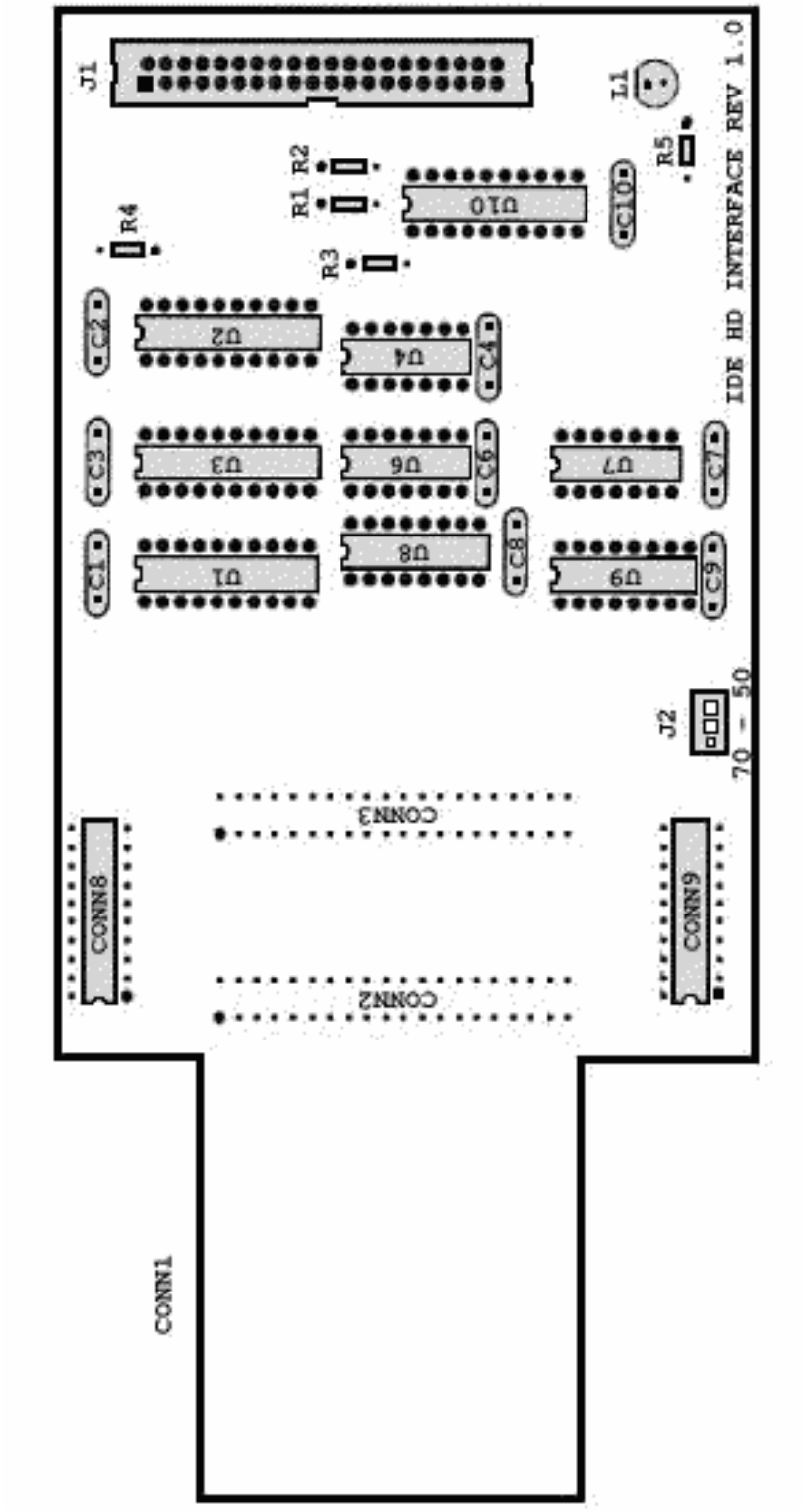
- 1 μF

The IDE Interface uses the following resistor value:

- 10 KΩ

E.3 PCB Layout

The PCB layout is included here as Figure E.3.



The PCB has one set of jumpers, J2, used to select the addresses at which the device is active. The options are:

- 1-2 closed, $FF7x ($FF70-$FF7F)
- 2-3 closed, $FF5x ($FF50-$FF5F)

The PCB has the following headers:

- L1: LED header (2-pin)
- J1: IDE connector (40-pin)

Finally, the PCB has several sets of options pads, CONN2, CONN3, CONN8, and CONN9, each supplied with plated-through holes. Two of these sets, CONN2 and CONN3, are designed to accept either header pins or a 40-position 0.100 centered card-edge connector. Both these options pads have the full 40-pin CoCo 3 expansion port bus brought out to them. (The CoCo 3 card edge connector does not supply the +12V and -12V lines – connectors 1 and 2 – that exist on a CoCo 1 or CoCo 2 card edge connector. These lines are not brought out on the CoCo IDE Interface Card.) If you solder a card-edge connector to either CONN2 or CONN3, you can plug a floppy controller into it and thus use the IDE Interface and a floppy controller without needing a Tandy MultiPak, SlotPak III or Y-cable. If you currently use a Tandy Multi-Pak or SlotPak III to supply +12V and -12V to an old floppy controller, you will continue to need to do so. The CoCo 3 does not supply these voltages, so the CoCo IDE Interface card cannot supply these voltages either.

It is possible that future cards will be designed to connect to Jy using header pins; one possible such use is a Fast-232 card, another is an autoboot PROM for use by OS-9. Thus, we suggest CONN3 be kept open for future expansion and that you use CONN2.

If you solder a card-edge connector in place, be careful to plug the cartridge PAK into it with the "top" side facing the CoCo. If you have a difficult time finding a 40-position 0.100 centered card-edge connector, you can take a standard ISA card connector and cut it down to two 20-position pieces. See figure XX.

The other two sets of option pads, CONN8 and CONN9, are designed to accept DIP chip sockets and are supplied with +5V and ground assuming a 20-pin DIP socket. These are provided for use by those brave enough to build additional features onto this IDE Interface.

E.4 Schematic

The schematic is included on the following pages.

Appendix F

Acknowledgements

The GCCC would like to acknowledge the following people, listed in alphabetical order, for their involvement in this project. They were instrumental in one or more of the following areas: design, layout, debugging, testing, software, documentation, and construction.

Carl Boll
Gene Brooks
Allen Dekok
Mark Farrell
Wes Gale
Brian Goers
Scott Griepentrog
Roger Hallman
Jim Hathaway
Brett Heath
Paul Jerkatis
Edward Kuns
Howard Lucky
Tony Podraza
Brian Schubring
George Schneeweiss
Bob Swoger
Justin Wagner

We are sure that we have forgotten someone. We hope that you will accept our apologies for this lapse, and if you let us know we will correct this error in later "printings" of this documentation. Of course, the people that helped build the boards at various GCCC meetings and special gatherings are too numerous to list.

LIMITED 30 DAY WARRANTY

The Glenside Color Computer Club, Inc. (a non-profit organization) warrants to the original buyer of this product that the hardware is free of defects in materials and workmanship for a period of 30 days from the date of receipt of this product from The Glenside Color Computer Club, Inc. If the product should prove not to be in good working order during the specified warranty period the Glenside Color Computer Club will, at its sole discretion, repair or replace the defective product.

The Glenside Color Computer Club, Inc. further warrants that the included software is as free of bugs and errors as we can make it.

If the product has been modified or the failure is due to obvious abuse, misuse or misapplication (to be determined by The Glenside Color Computer Club) the warranty is void and The Glenside Color Computer Club has no obligation to repair or replace the product.

The customer is responsible for properly packing the defective product, shipping the product and the cost of shipping the product to the specified repair facility. The Glenside Color Computer Club, Inc. will ship the product back via a carrier at no cost to the customer. The carrier and method of shipment will be solely at the discretion of The Glenside Color Computer Club, Inc.

Before returning the product for repair or replacement the customer must first get an RMA (return merchandise authorization) from the project coordinator. To get the RMA send E-Mail to: root@chicoco.chi.il.us or a letter to: The IDE Project C/O Carl Boll 6242 S. Menard Ave. Chicago, IL 60638

Under no circumstances will The Glenside Color Computer Club, Inc. be liable for any direct, indirect, consequential or incidental damages arising from the use or inability to use this product or its accompanying software.

The Glenside Color Computer Club does not warrant that the optional pads provided for experimental or expanded use will be usable on all Tandy Color Computers.

The Glenside Color Computer Club does not warrant that this product will coexist with all other hardware available for the Tandy Color Computer.

The Glenside Color Computer Club does not warrant this product to work with any devices other than ATA-2 compliant hard drives, please see the instruction manual for further clarification.

The Glenside Color Computer Club will not be liable for the product if it doesn't work with a specific computer but does work on our test platform. Some Tandy Color Computers are not built to specifications and we can not warrant that our product will work with them.