Hard Disk Controller: the Disk Drive's Brain and Body

James Jeppesen; Walt Allen; Steve Anderson; Michael Pilsl
Infineon Technologies; Memory Products, Network and Computer Storage
600 South Airport Road; Suite #205; Longmont, Colorado USA
(James.Jeppesen; Walt.Allen; Steve.Anderson; Michael.Pilsl)@infineon.com

Abstract

Integration of the Hard Disk Controller (HDC) today has taken on an extensive amount of functionality. From the host interface, error correction code, disk sequencer, microprocessor(s), servo control logic, buffer controller, to the embedded memory, the HDC has become a true system on a chip. Depending on the product, embedded DRAM is used as buffering for data between the host and media and possibly for storing controller firmware. By bringing all these blocks into one chip, pin counts can be reduced and higher data flow speeds can be obtained by decreasing the interconnect delays. However, the challenge for designers is in test and verification of the design during development and production.

1. Introduction

Computer mass storage devices once took up an area about the size of a 2-drawer legal file cabinet back in the 1980's. A current drive produced for digital cameras is smaller than a credit card, about the depth of a 3 ½ inch floppy, and holds at least 10 times the data—all at a reduced cost. In 1995, a ~400 MB drive cost nearly $500. Now, a 20 GB drive can be bought for only $150. What has enabled the manufacturers to achieve such changes? One area is system integration.

The hard disk controller (HDC) has experienced rapid integration as driven by cost reductions. Moving from a six-chip design, the current HDC solution is one chip. Future products will integrate the read/write channel into the HDC further reducing the total number of parts the hard disk drive (HDD) will require to perform its function. The paper presented here will look at the current scale of integration and the testability issues created by this integration.

2. HDC Functional Block Description

The Hard Disk Controller today is made up of the following common blocks: Host Interface, Buffer Controller, Disk Sequencer, Error Correction Code (ECC), Servo Control, Central Processing Unit (CPU), Buffer Memory, and CPU memory. See Figure 1 for a block diagram of a disk drive.

2.1. Host Interface

The host interface's primary function is to provide a standard protocol for the Disk Drive to talk to a host system. The host system can be anything from a server or PC to a simple peripheral or consumer product--such as a printer or digital camera. There exists a number of distinct protocols in the disk drive industry for performing this link. Some of the major interfaces are ATA, SCSI, and Serial interfaces. The main driver between selecting which interface to use is cost to performance. The design trend for the host interface is to have multiple interface blocks. Each block supports a particular host protocol and has a standard back end interface to the rest of the HDC. This allows for maximum synergy among designs. Depending on the interface chosen and the performance desired, the size of the host interface can vary dramatically from a few thousand gates to over a hundred thousand gates. Currently, the host interface is the only block that is covered by any kind of published industry standard--albeit many different public standards. These standards define physical transfers, required registers, and command sets.

2.2. Buffer Controller

The main function of the buffer controller is to provide arbitration and raw signal control to the bank of buffer memory. This memory can be Static Random Access Memory (SRAM), Dynamic Random Access Memory (DRAM), or Embedded memory. Typically the host interface, disk sequencer, ECC, and CPU all need access to this buffer memory. The buffer controller, under some priority scheme, will prevent these blocks from colliding while accessing the memory buffer. This block may also contain logic to help with the automation of moving data to and from the host. The size of this block can vary depending on the number of memory configurations supported by a single controller. The system throughput and performance can be greatly affected by this block.

0-7695-1200-3/01$10.00 © 2001 IEEE

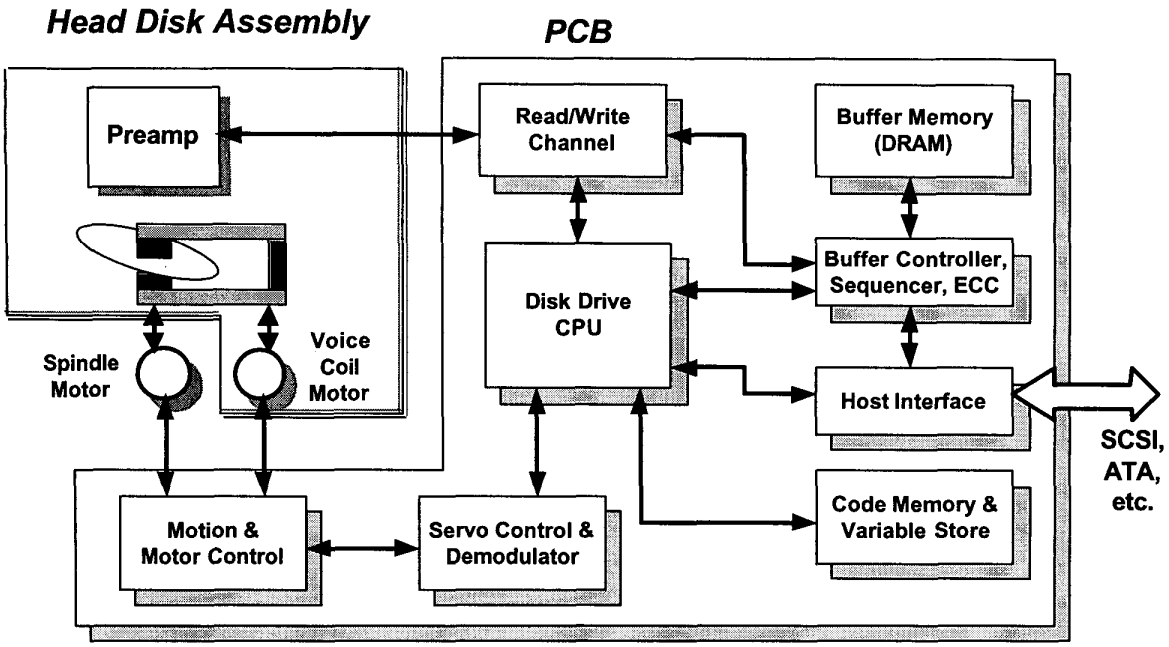

Figure 1. Hard disk drive subsystem

2.3. Disk Sequencer

The main task of the disk sequencer is to oversee and manage the transfer of data between the disk interface (referred to in this paper as the NRZ pins) and the data buffer. The term, NRZ, is defined as non-return to zero. The NRZ bus has become a normal name for the 8-bit data interface between the read/write channel and the HDC. For a disk write operation, the disk sequencer takes user data, appends additional fields such as ECC bytes, and writes out the newly formatted data to the media interface through the NRZ pins. (NRZ is defined as non-return to zero.) Conversely for a disk read operation, the disk sequencer reads formatted data from the NRZ pins and converts it back into user data that is then sent to the host interface. The process is complicated by the fact that disk media has servo wedges written on it that cannot be over written by user data. (Servo wedges, typically 50-80 per revolution, contain gain control information for the read/write channel; cylinder/track location information; and head alignment patterns. The servo wedges are written at the factory by special equipment called, "Servo Writers.") The user data sometimes must be split on either side of these servo wedges. The disk sequencer handles the split data fields for both read and write operations. Furthermore, since the spinning disk cannot be throttled, the data rate from and to the disk must remain constant. The sequencer is in charge of pulling data from registers, data buffers, and the ECC block at precise times in a disk write operation, and in charge of sending the NRZ data to the correct blocks during a disk read operation. The disk sequencer is often referred to as the disk formatter.

2.4. ECC (Error Correction Code)

In terms of a single function, the ECC is one of the largest blocks of an HDC controller. This block is responsible for appending ECC symbols to the user data and also to check and, if needed, correct the user data before it is returned to the host. The ECC size is greatly affected by the chosen correction capable of the hardware and the amount of software intervention required to perform the correction. Along with ECC syndromes, this block often contains some type of Cyclic Redundancy Check (CRC) function to keep the probability of miscorrection to an acceptably low level. Current disk drives are specifying a nonrecoverable read error of 1 in 10^{14} bits read [2].

2.5. Servo Control

The servo control block has a lot of different definitions depending on the particular implementation. In this paper,

the servo block refers to general logic used in aiding the spinning of the discs and in the positioning of the actuator on the disc. It does not refer to the power devices necessary to drive the spindle motors. This block is uniquely customized to the particular customer's strategy for motion control of the Head Disk Assembly (HDA). Therefore, it is difficult to standardize and thus lends itself to an Application Specific Standard Product (ASSP) strategy.

2.6. CPU (Central Processing Unit(s))

The CPU of the HDC can be implemented in multiple ways. Single 8-bit, 16-bit, or 32-bit microprocessors and Digital Signal Processors (DSP) have been used, as well as combinations of these cores. These cores are used to control the overall system or may have a very specific task. The CPU has the highest gate count of all the logic blocks except memory. It is also the most complex from a simulation and integration standpoint.

2.7. Buffer Memory

The buffer memory is used as a temporary storage of the user's data either on its way to the disc or returning to the host. This memory may also serve as variable storage or even code execution space for the CPU. This memory traditionally has been made up of both SRAM and DRAM. The buffer memory can be either external or embedded within the HDC. By embedding the memory, a higher throughput to the buffer is possible. However, embedding the memory increases the integration and testing difficulties.

2.8. CPU Memory

The CPU memory can be made up of Read Only Memory (ROM), SRAMs, Flash, or DRAMs. Also, any combination of these memory types can be utilized in the product design. This memory is where the op-codes for the CPU are stored for execution. Currently, some sort of non-volatile memory is required. However, the use of volatile memory to supplement the non-volatile memory is becoming standard. The benefit of this volatile memory is that it can be changed often, execution performance increased, and manufacturing costs reduced. If volatile memory is available, it quite often doubles as code variable storage for the CPU.

3. The Challenge of Integration

Integration of more and more HDC functional blocks into a single chip is not without issues. While integration can lead to cost savings, if not managed appropriately, it can lead to cost increases, or even failure of the project.

As each module was integrated over the years, the externally observable connections were no longer available for test. This forced the designer to find alternative methods for testability without adding significant cost to the final product. The testability issues have different objectives: development tests for the designer to prove function; wafer tests which ensure the die was successfully manufactured; final assembly tests to verify the customer receives a functional part; and finally a means for the customer to develop and debug application firmware/software on the embedded microprocessor. We will now look at the process for developing the product in greater detail and each of the testability objectives.

3.1. Development Phase

The development phase sets the objectives for the project. Major concerns include team make-up, features required for the product, and test methodology from both a producer and customer viewpoint.

Team make-up has implications on the overall design methodology. If multi-sites are involved in the design of the project, functions must be carefully chosen to streamline the utilization of the resources within each site. Often, more than one geographical site is utilized, which may or may not extend across international borders, in the development of products. To control the design process, version control packages track the updates to the design as each designer releases their module for general use. Without version control, an accidental bug placed into the design can take many man-hours to locate and correct, impacting all engineers on the team.

The definition of the product takes on a new dimension in an integrated part. Defining the features includes package selection and testing, which are all required before design can begin. How is this different from the non-integrated process? Package selection impacts the number of pins available for functional and testing use. Heat dissipation concerns limit features based on cost of packages needed to handle the thermal load. The non-integrated design could use a chip clip and a logic analyzer as a testing plan. Of course, the integrated part has most module interconnects hidden from the outside world. (We will look at testing issues later in this section.)

Once the definition and design has been completed, the conversion to hardware begins. Most of us have seen HDL to netlist flows. Highlighting some important aspects of the flow shown in Figure 2, static timing analysis (STA) and formal verification have become a very important step in the

development of the highly integrated HDC. STA is a fast method of inspecting the inter-module and intra-module timings, clock skew, and test mode timings created during synthesis. From STA, more exact timing constraints can be produced to improve the synthesis of the netlist during subsequent synthesis operations. Formal verification ensures the HDL code matches the netlist created. Formal verification is very useful as patches to the netlist and HDL occur. Once the netlist is laid out and true timings are available, another flow is followed to verify the results.

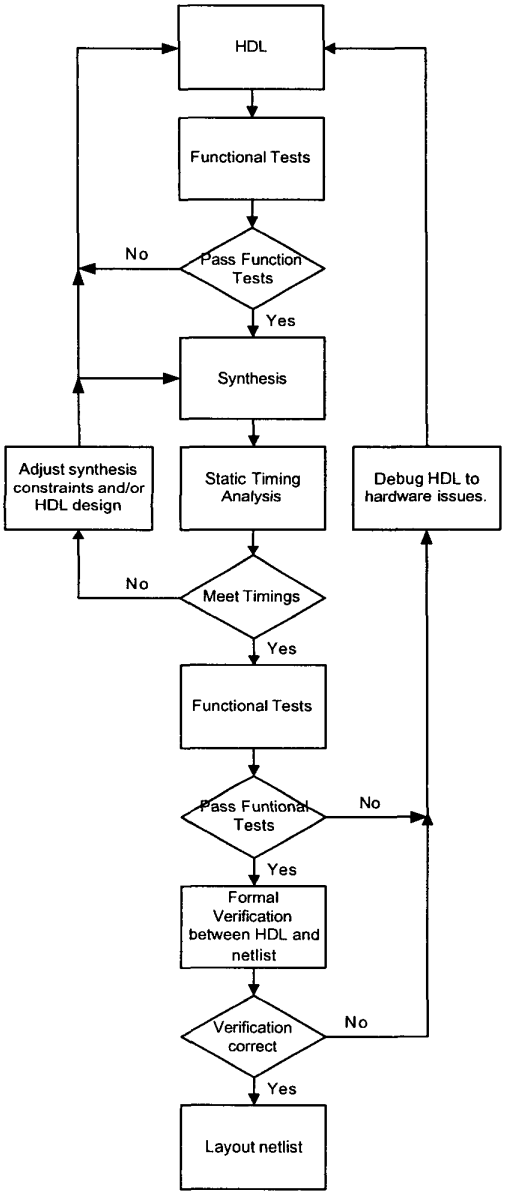

Figure 2. HDL to netlist flow.

In the layout flow shown in Figure 3, STA and formal verification are the main tools used to ensure the layout meets the design function and timing requirements. Since STA and formal verification are performed, the running of functional tests is not really necessary to prove the layout in design terms. However, one should perform these tests as an extra measure of confidence.

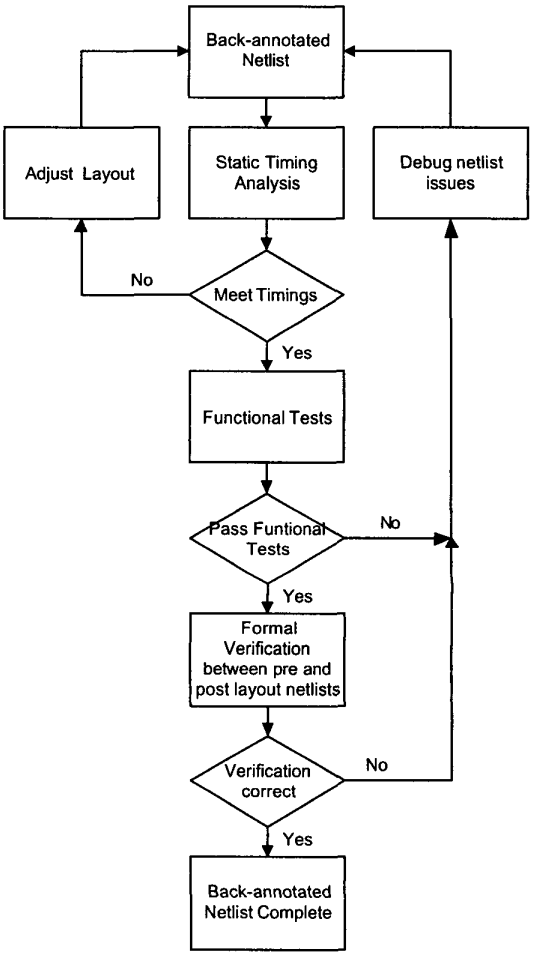

Figure 3. Layout to release flow

3.2. Test point of view

As referred to earlier in this paper, testability issues are a main part of the development phase. Three points of view need to be considered when planning testability. First, the design engineers must formulate tests to verify the functionality of the chip. Next, test engineers need wafer and production tests to verify that the silicon was manufactured correctly. Finally, the customer's viewpoint needs to be considered--both for demonstrating a quality

working part in the disk drive system and for creation and debug of application firmware. Each viewpoint must be addressed to deliver a product that meets or exceeds the customer's expectations in terms of quality and ease of application.

The designer of each module within the HDC creates tests that verify the functional aspect of their design. These functional tests are typically created for use in a simulation environment. However, many of these functional tests have been converted to run on a production tester. Thus, the designer must keep in mind only pins exposed on the package should be used in the functional tests created. Special test modes can be added to the design that change the configuration of the pins to expose normally buried signals. Care should be taken to minimize the creation of these modes. Each special start up required to enter test modes takes up time on the production tester. Production test time is a major factor in final part cost.

The next viewpoint addressed is the production tester. Once the design has been transferred to a wafer, patterns are run to determine which dies have been successfully produced before packaging. Wafer tests may include scan vectors, functional vectors, IDDQ current test, and built in self-tests (BIST). Scan vectors are generated through automatic test pattern generation (ATPG) software tools. If the designer has created the design with scanability in mind, over 90% of the logic can be tested through the ATPG patterns.

Once packaged, the viewpoint changes to delivering a quality product to the customer. As stated earlier, the designer must have thought through test methodologies to catch parts that are marginal or damaged through the packaging process. The designer must also consider the application in which the product is intended to function. The designer needs to add features to assist firmware/software engineers in developing the HDC into the final disk drive. Let's consider these closer.

Even though a die has passed the test vectors while part of a wafer, a die may have marginal characteristics that will cause it to fail under stress. A standard method for creating the stress is the burn in process. Special tests are created to exercise the part during the burn in process. The object of these tests is to exercise as much logic as possible while the part is exposed to the stress. Once the burn in process is complete, tests are run to remove parts that suffered from infant-mortality (failures caused by the stress).

From the viewpoint of the customer utilizing the packaged part, one of the first problematic items to become integrated was the microprocessor. As long as the external memory bus is brought out to an external ROM device of some sort, the troubleshooting issues are minimized because the primary interface and partitioning of logic is viewable, probe-able, and therefore analyzable. Now real problems start when any sort of memory, such as embedded-code SRAM, is pulled into the device, and for pin savings, or timing margin, the designer chooses to eliminate this visibility. This visibility issue pertains to every major integration carried out in the modern system on a chip. With integration, the complexity increases, the visibility decreases, and system debug truly becomes the nightmare of the firmware/systems/test engineers.

Now that we have identified a root drawback to highly-integrated chips, from the troubleshooting aspect, what can we do about it? One familiar approach is to apply Built-In-Self-Test (BIST) to memory devices. On the old non-integrated system, if one wrote an AAh to a memory device, then read back an A8h, the faulty bit of the memory device or CPU chip is clearly evident. When it is shown that the A8h is evident at the pin of the memory device, replace the device. When the BIST fails, the memory or the BIST logic has a problem. If the BIST passes, and the CPU still fetches bad data, interface logic should be examined. One drawback is that pattern sensitivities may exist with patterns that are not tested in the BIST. This results in a BIST pass condition, while system execution fails.

Another employed technique is the implementation of a test multiplexor. This logic dedicates a small number of pins to the presentation of a predetermined subset of important internal signals. This is extensively used in testers to increase test coverage, while decreasing test time. However, if systems/firmware engineers are incorporated in the process of choosing these signals, critical system debug visibility can be achieved as well. Drawbacks include not having the necessary signals to view the particular bug of the day, as well as having signals that need to be examined together that have different select settings for the multiplexor. Keep in mind, there can be hundreds of internal signals of interest, yet only a handful of pins for these signals to be multiplexed out upon. Therefore, it is imperative to plan this test strategy carefully.

One of the simplest items to implement is increased register visibility from the CPU, by the firmware. Unfortunately, many chips today are created with write-only registers. This hinders firmware creation and debug, while only saving minimal logic area. Another method to increase visibility is to insure that FIFOs are readable from first cell to last cell. As FIFOs are often used as an important part of the partitioning interface between logic blocks, this functionality can be used to isolate faulty logic to the source block. Visibility of state machine states can be a reward, as well. While understanding a CPU might not have the resolution to capture every phase, it can show when state machines receive unexpected states and end up in unexpected places. Drawbacks mainly include the increased code space and

execution time, which alter the behavior of many real-time systems. For debugging time-critical functional areas, this technique generally should not be applied.

If a design is core limited, extra signals can be brought out to supplementary I/O pads. A secondary test package can be chosen to present these signals to the outside world. Once debug is complete, the original production package can be shipped with the original die. If time and cost constraints are met, this can be an extremely low risk methodology of increasing visibility. Along these lines, even special ball grid array (BGA) substrates have been produced to bring out microprocessor trace functionality, creating the visibility to track CPU flow and op-code execution results.

On-Chip Debug Systems (OCDS) are becoming available to manage the embedded processor visibility problems. Some features existing today include on-the-fly access to any memory location or internal register, real-time hardware breakpoints, virtually unlimited software breakpoints, and more. If your system includes this capability, requirements for external memory bus visibility becomes a low priority, in most cases.

Co-verification, the utilization of firmware and hardware in a system environment, is growing in popularity. The impetus comes primarily in making parallel, the current serial cycle of developing hardware and then developing firmware, in an effort to further reduce development times. While this has been shown to reduce development times in complex/expensive efforts, it should be viewed as another mechanism for creating visibility of all nodes in an integrated environment--nodes that just could not be seen any other way.

Even with these techniques at hand, management should schedule sufficient time for increased testing and debugging. No longer can the engineer or technician just place a test clip on the offending device and "see the bug". Many of these techniques, can only be applied in conjunction with writing specific code segments. This process alone increases time. Techniques such as co-verification increase testing time and must be managed.

Increasing utilization of these techniques increases visibility; yet recoup costs, by savings in pins, packages, and potential silicon spins.

4. Conclusion

Integration has brought the price of the hard disk drive down for the consumer. This paper has shown how designers have dealt with the integration in terms of testability and design flow.

As integration continues to bring more functionality into the HDC, designers will need to manage the testability issue. New methods for tackling the three viewpoints (designer, production, and customer) are waiting to be found.

5. Acknowledgement

The authors wish to express our thanks to Chuck Gravelle for his editing efforts and Donna Apel for her help in generating figures used in this paper.

6. Literature

[1] M. Pilsl and B. Rohfleisch (1999), "Embedded DRAMs for Hard Disk Drive (HDD) Controllers", *DAC Tutorial Presentation '99.*

[2] Seagate Technology, *Barracuda ATA II Family Product Manual, Rev. B,* Seagate Technology, May 2000.