Understanding Secure Digital I/O Performance in Systems and Cards

Seung Yi

Chief Technologist

Codetelligence, Inc.

March 15, 2004

1.0 Introduction

The proliferation of feature-rich mobile devices, with ever-increasing processing capabilities has created a desire for expansion options previously seen only in the laptop/desktop space. Consumers are increasingly demanding expansion options to increase capacity (storage) or capabilities (features) in order to fend off, for the moment, the threat of obsolescence and future (largely unknown) needs. Multimedia content has been the driving force in the industry to create the necessary standards and technology in audio players, digital cameras, personal digital assistants and smart phones. Expansion began with memory storage in the form of memory cards (with a plethora of existing standards) utilizing low cost NAND flash technology. As mobile devices become increasingly more powerful the line between a single function consumer electronic device and a more general computing device becomes blurred. One indication of this trend has been in the standardization of I/O expansion in mobile devices.

Previously I/O expansion in mobile devices consisted of proprietary hardware/software interfaces, typically only available to the specific manufacturer and in many cases to a specific device model. Soon afterwards a generation of devices adapted technology from the realm of laptop computing in the form of CompactFlash I/O. CompactFlash I/O expansion offered a relatively painless way to incorporate PCMCIA standards to a more mobile-friendly form factor. Demand for thinner and lighter devices has pushed the need for even smaller I/O expansion technologies.

Enter Secure Digital I/O (SDIO), introduced in the later part of 2001, as an evolutionary offshoot of Secure Digital Memory technology. SDIO is based on similar electronics and mechanicals borrowed from the original SD memory card specification, and offers device manufacturers a convenient way to support memory and I/O cards in the same slot. SDIO extends the specification to meet I/O specific requirements in terms of power, plug and play, and I/O commands and signals. The addition of I/O expansion capability to an SD memory slot can be implemented at minimal incremental cost, largely due to the availability of more capable controllers.

This paper describes some of the performance issues that should be taken into consideration when designing SDIO capable systems and cards. Design issues can be encountered on both host and card side implementations, from silicon to software. This paper provides a general overview of SD Memory and SDIO and then goes on to describe some application specific and common issues encountered in the design of such systems. The information presented here is based on experience in system software design for various SDIO capable platforms including software architecture, silicon-specific host drivers and card-specific control software.

2.0 Overview of SD/MMC Memory

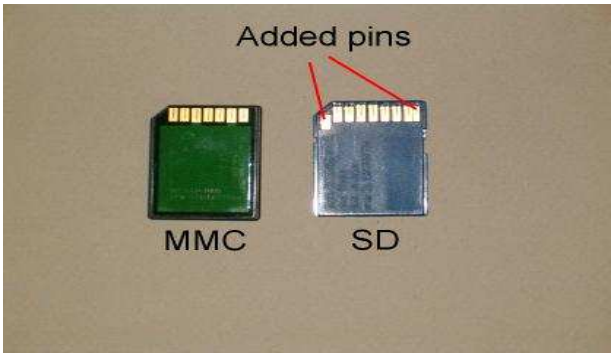
Secure Digital I/O has its roots in Secure Digital Memory and Multimedia Memory card technology. To understand some of the features and extensions offered by the SDIO standard, one must examine the earliest incarnations of this card format.

The Multimedia Memory Card (MMC) format was designed as a low pin count, low power, thin profile non-volatile storage card standard. The electrical interface is a simple direct CMOS/TTL logic-drive serial bus based on two open synchronous serial protocols: the industry standard Serial Peripheral Interface (SPI) and the new MultiMedia Card native interface. All MMC cards must support both protocols. The intent for having both protocols was to initially design card readers and mobile devices using pre-existing off-the-shelf controllers supporting SPI mode. SPI however is a non-optimal implementation as many SPI controllers are limited in speed (~2-8 mhz) and capability (e.g. no hardware CRC generation). SPI lacks any built-in data read/write flow control and requires additional software to manage simple data transfers. The MMC native interface solves many of these problems by requiring controllers to have built-in CRC generation and data flow control (read start and write acknowledgement bits). The data rate was increased to support clock rates up to 20 Mhz, translating into an effective transfer bit rate of 20 Mbs/sec. MMC in SPI and native modes use 7 pins and the SPI signals are shared with the native signals. A device can be placed in SPI mode when a special command is issued while the SPI chip select pin signal is driven low.

The 1-bit native mode uses a separate bi-directional channel for MMC commands (i.e. read, write, card address) in conjunction with a separate 1-bit channel for data transfers. The communications protocol is message-based consisting of a 48-bit command message, acknowledged by a response message of varying length (48-136 bits). If a command includes a data portion (i.e read or write command), the data payload is sent over the 1 bit data channel (SPI data transfers uses the same pins for command/response and data). All commands, responses and data are protected by a CRC code appended to the end of each frame. For the purpose of this paper, electrical and timing details and command/response formats are omitted. You can obtain specifications from www.mmca.org and www.sdcard.org for more information.

The Secure Digital Memory format extends the MMC specification for higher performance and SDMI (Secure Digital Music Initiative) conformance. Secure digital memory still supports SPI and MMC 1-bit native electrical interfaces, but adds a higher performance 4-bit interface at 25 Mhz. This effectively provides 100 Mb/sec transfer rates. Secure Digital Memory cards are slightly thicker and provide a mechanical write protect switch (non-electrical). SD Memory card pins share the same 7 pins with MMC cards with the addition of 2 pins to support the 4-bit interface. SD and MMC cards can use a single card slot that supports both pin arrangements. SD memory cards can be used in systems with legacy SPI and MMC controllers, however they will operate with lower performance. Software handles the differences in the use of each card type.

The following image shows the card pin/pad layout with the additional pins on the SD card highlighted:



The following image shows the relative size of various SD/MMC/SDIO cards. The bluetooth and WLAN cards are courtesy of Socket Communications Inc.

2.1 Bus Topology

SD Memory and MMC card slots can be implemented in a bus signal topology or a point-to-point topology. A bus arrangement allows a single controller based system to control multiple cards at a reduced cost. Note that in a bus topology where SPI is used, separate chip selects are used to address each card. In native modes (1 or 4 bit SD), cards are addressed using a command packet instead of individual chip select signals. Each SD and MMC card is assigned a relative card address. Cards can be selected/de-selected using a special command and its relative card address. Deselected cards remain idle and do not respond to SD/MMC commands.

Bussed topologies have inherent problems electrically and functionally. Electrically, hot swapping cards while device signals are bussed together can inject errors onto the bus. Functionally, a high performance card may be forced to operate at a reduced rate based on the slowest clocked card or the card whose interface can be supported simultaneously

by the controller. For example, an SD card used in a bussed system with an MMC card may be forced to operate in 1-bit mode and at a 20 Mhz frequency. Constant selecting and deselecting of cards when applications must access multiple cards simultaneously may also become a performance bottleneck. For example, take the case of an SDIO WLAN card used to download content to an SD Memory card on the same bus. Each transaction would require the system to issue many CMD7 messages to select/deselect cards as commands are issued from both drivers simultaneously, effectively cutting the bandwidth by up to 10%, depending on the transaction being performed.

Whenever possible, avoid operating SDIO in a bussed topology. The optimal mode of operation for SDIO is the SD native interface, which offers the best performance and lowest processing overhead in most designs.

2.2 Secure Digital Music Initiative

Secure Digital memory cards were designed to support SDMI for the security and rights management for digital content (primarily music). The technology used to encrypt and authenticate the data is based on methods from the 4C Entity (http://www.4centity.com/). The standards describe content protection techniques for the transfer of digital media to and from SD memory cards. SD memory cards can be partitioned into secure and non-secure areas for this method. Secure area access requires proper authentication between the card and the SDMI compliant application. 4C content protection also encrypts the SD data stream to prevent bus snooping. Only a trusted application can decrypt the data stream transferred from the secure area of the SD memory card. The SD controller hardware does not require any special encryption/decryption blocks as the payload of secure SD reads/writes is of no interest to the controller but of use only to a trusted SDMI application. Hardware accelerators can reduce the CPU burden when encrypting/decrypting the SD transport payload, however the content itself may have its own non-SD related encryption/decryption requirements based on the content source and type making the hardware accelerator unusable in this situation.

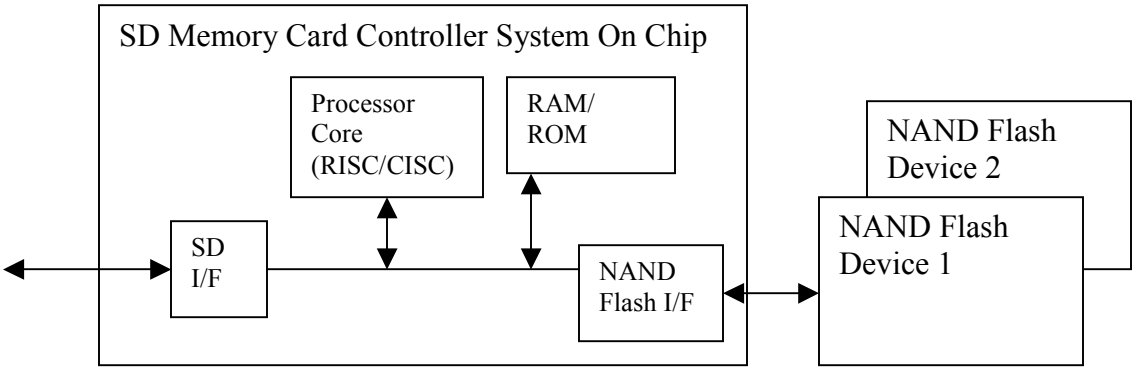
2.3 Flash Storage Concepts

Secure Digital and MMC cards utilize solid-state memory, typically NAND flash memory. Some cards can also employ write-once memories such as masked or one-time-programmable devices. NAND flash is a low cost storage technology with high-speed page erase-rewrite cycles and relatively fast read cycles. NAND flash is suited for data storage as opposed to code execution due to its simple electrical interface and paging mechanisms. NAND flash utilize a low pin count multiplexed interface and high-speed paging suitable for block-mode access (i.e. filesystems) but unusable for random byte/word access for code execution. NAND flash arrays are arranged in pages (blocks), which can be designed much easier with multiple redundant blocks on the same silicon die. Bad blocks can be “trimmed” away and redundant ones activated. Bad block data can be stored on the die so that an external controller can use this information to steer

data around bad blocks. All these techniques can greatly improve the die yield, resulting in lower costs.

Flash memory devices have a limited number of reliable write/erase cycles. Once a cell has exceeded its write endurance rating, it may fail to program or erase. Typical flash devices have erase/write endurances of 10K cycles or more. By spreading the write/erase cycle among all the pages in a NAND flash device, the longevity of a device can be improved dramatically and can even exceed that of traditional magnetic-media based hard-drives. The technique of distributing erase/write operations throughout the NAND flash device is called wear leveling. SD and MMC devices require no erase operations (even though the protocol defines an erase command) because the operation is handled internally on the card in conjunction with wear leveling. A write operation to a NAND device would require the host to read the block, modify the block in ram, erase the block on the device and then write back the new block. Fortunately for system designers, block erase and wear leveling operations are "hidden" within the card. SD/MMC cards appear to a system as a large block-addressable array of bytes. This greatly simplifies software implementations as complex wear leveling and device-specific erase operations are not required by the host. Many flash storage devices appear to operating systems as rewrite-able block devices. This does however require a fairly complex controller to hide the underlying NAND flash design.

The following diagram illustrates a typical memory card system.



Most SD cards (and other flash card standards) utilize a highly integrated system on chip (SOC) consisting of a processor core, RAM/ROM, SD and NAND interfaces. The controller processes SD commands and performs translation, wear leveling tasks and outputs the appropriate read/erase/write commands to the NAND flash array. The controller utilizes embedded software stored in ROM to perform the complex wear leveling algorithm and bad block steering. Typically a manufacturer uses one controller for many different card sizes as the internal software can determine the array size from the populated NAND flash devices.

Not all NAND flash devices have the most convenient page size for erase operations. Since a block operation must erase the entire block and re-write it with the new data, a file system that uses small block sizes relative to the card's "suggested" erase block size may force multiple internal erase operations on the same block when updating parts of a file.

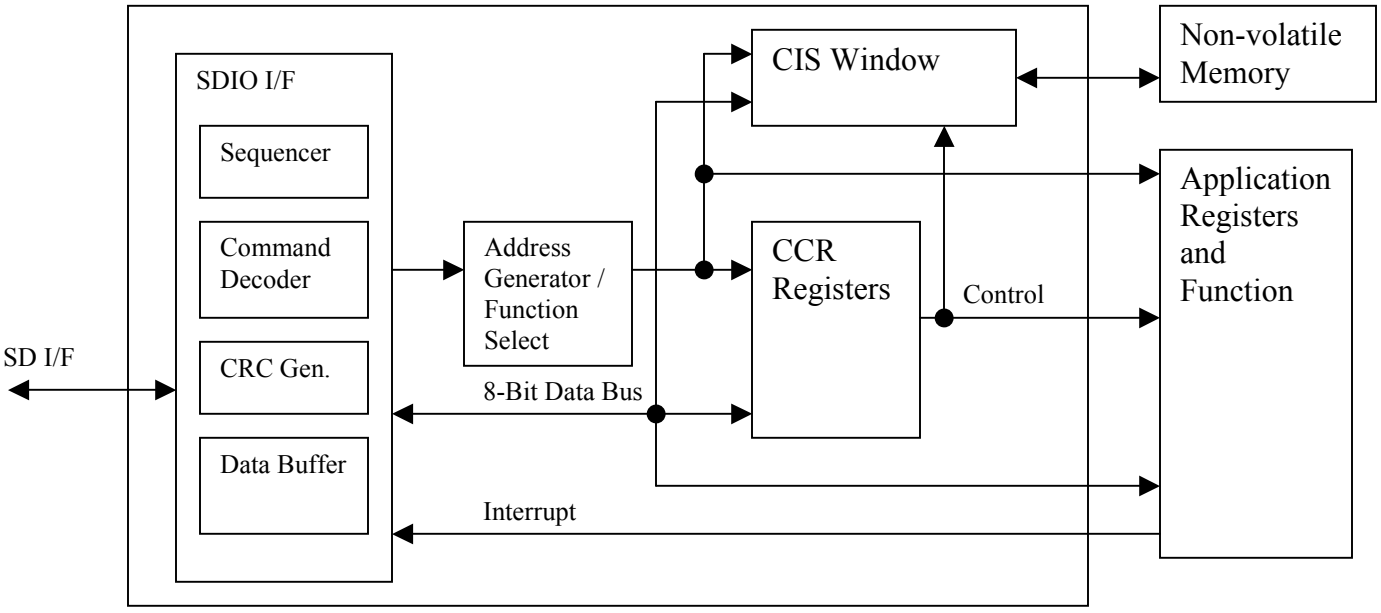
A designer should be aware that each SD read/write request requires the controller to perform the necessary logical to physical block translation. Write operations also entail the erasure and reprogramming of a block. By maximizing the amount of data from the read/write request, the overhead (translation, erasure, writing) associated with the request becomes less significant. Optimizations can be made to how the filesystem and/or applications use the card to get the best read/write performance.

3.0 Overview of Secure Digital I/O

The Secure Digital I/O standard extends SD memory capable systems to also support I/O peripherals such as bar code scanners, cameras, wireless adapters and GPS modules. The standard uses the same 9-pin slot connector and slot mechanics and defines new I/O signaling protocols over existing pins. The SDIO card form factor allows for an extended region to house electronics, RF components, camera lens assemblies, or connectors/antennas. Fundamentally SDIO use the same transport electrical interface, SPI or SD native with the addition of a few new I/O specific commands and signaling mechanisms. To meet the requirements of asynchronous signaling, the standard introduces I/O signaling for Interrupts and data read suspending (Read/Wait). The SDIO standard contains simple, elementary changes allowing existing SD/MMC controllers to be retrofitted into functional SDIO controllers with some "acceptable" limitations. Some of the limitations of retrofitted controllers are described section 5.0.

SDIO defines the same SD base clock rate of 25 Mhz and uses 1 or 4 data pins. SDIO also defines a low speed card type that uses a 400 Khz clock and a fixed 1-bit mode of operation (4-bit is optional). SDIO bus topology is similar to SD and many of the signals can be bussed, however due to complexities that interrupt signaling, read wait protocol and I/O clock matching introduce, it is highly recommended that the SDIO card slots be separated into a point-to-point topology.

The following diagram shows a possible SDIO card-side controller implementation.



3.1 Register Mapping Model

SDIO creates a register level view of an SDIO function or multiple functions through a set of common registers and application specific registers. A card has one to eight (numbered 0-7) functions with function 0 reserved for common configuration. Each function is assigned a 128 Kbyte register address space for configuration and operation. The common control register (CCR) set is implemented on every SDIO card and resides in the card address space for function 0. An SD host configures various operational parameters of the SDIO card through the common register set. A card containing one or more functions may have separate software modules (drivers) that operate on registers located in their specific address space. The registers are accessed byte wide using the SDIO read/write register command (CMD52). It is tempting to view SDIO as a parallel-to-serial I/O bus where an SDIO device can be locally mapped to the system address space. This however is not the case in practice. All known SDIO controllers treat SDIO commands like traditional message-based SD commands, that is, they are packets sent and received via a serial transfer engine. This engine may require an interrupt per command, unless software polling is used. A CMD52 packet requires minimally 96 bits (includes command and response) of information for just reading or writing one 8-bit register. The serious limitation encountered is that the overhead for each register read/write is enormous compared to a true memory mapped bus. Instead of measuring read/write cycles in the 10s of nanoseconds, the cycles are extended into the 10s of microseconds.

 The SDIO register mapping strategy was designed to simplify infrequent configuration tasks and not to be used extensively for data transfer. Some devices such as slow I/O devices (low speed UARTs) can be used in this fashion with "acceptable" performance.

3.2 Interrupt Signaling

I/O peripherals typically require an interrupt to signal a host to attend to some event on a device. This may be the result of data arriving from a communications port or the completion of an operation. Polling can be employed where the data may be infrequent or performance is not an issue. Polling however does have a power consequence in that transactions are continually sent to the device even when the device may not have any useful data to return. Polling software often defeats power saving techniques that can place the processor into sleep/snooze mode within idle periods.

SDIO multiplexes a level sensitive interrupt signal onto the 3rd data line. In 1-bit data modes of operation this signal is non-multiplexed and fully asynchronous to the clock (i.e. requires no SD clock for sampling). In 4-bit mode, this signal however is a shared signal and the controllers on both ends (host and card controllers) are monitoring/driving

the 3rd data line continuously and altering behavior based on bus activity. The signal is sampled on each SD clock during zero-data periods and during gaps between multi-block data transactions. The consequence of this is the continuous application of the SD clock, which can also impact battery life. As a power saving measure, card software can typically request the host system to change the interface mode to 1-bit, remove the SD clock and operate interrupts completely asynchronously. When a large amount of data is to be transferred the card control software can request the interface to switch back into 4-bit mode. Switching interface modes incurs the overhead of a CMD52 bus transaction to update the card interface register. That latency maybe significant and should not be ignored. Switching modes requires careful card management to make ensure card states are in the correct mode to optimize power and response time. A discussion on host controller clocking is provided in section 5.0.

3.3 Plug and Play

SDIO is a "plug and play" architecture with methods for an operating system to load appropriate control software based on information located in the card's common register set or Card Information Structure (CIS). The CIS is a non-volatile memory region, accessible byte-wide, containing card or function specific meta-data. Each device has a common CIS along with one function specific CIS for each function on the card. A plug and play manager may be interested in the common CIS for device class, manufacturer ID, or power information, while a function driver may be interested in the function CIS for application specific parameters (i.e. MAC address, function capabilities). Control software can also be loaded into non-volatile memory on the card (beyond the CIS), however due to the different operating systems that a card may encounter, this feature is rarely, if ever used. SDIO defines standard device classes that expose a known register set and card behavior for control software (drivers). An operating system can support standard devices from a range of card manufacturers using a single class driver. Some class drivers include GPS, Bluetooth and still-image digital cameras.

3.4 Multi-function and Combo Cards

Increasing levels of integration will usher in a generation of SDIO cards that will include multiple discrete functions. A wireless card may implement multiple wireless protocols (GPRS, WLAN or Bluetooth) with each radio existing as a separate I/O function utilizing separate control software (sometimes from different vendors). Multifunction cards must be managed by system software that provides the necessary multiplexing and scheduling of SDIO requests. Some performance degradation is to be expected since the single SDIO interface is now shared among multiple I/O functions. SDIO does not use an I/O interrupt priority mechanism nor does it have any means of internally managing bandwidth. This complexity is left to system software. An SDIO command must in effect wait for the previous outstanding request to complete before a new request can be submitted. The SDIO standard does however define an I/O suspending protocol to allow preemption of current bus activity for a higher priority request. Unfortunately, many

cards and host controllers have yet to fully support this feature. The specification allows a data transfer on a specific function to suspend allowing another higher priority data transfer to occur on a different function. The suspended function's data transfer can then be resumed at a later time.

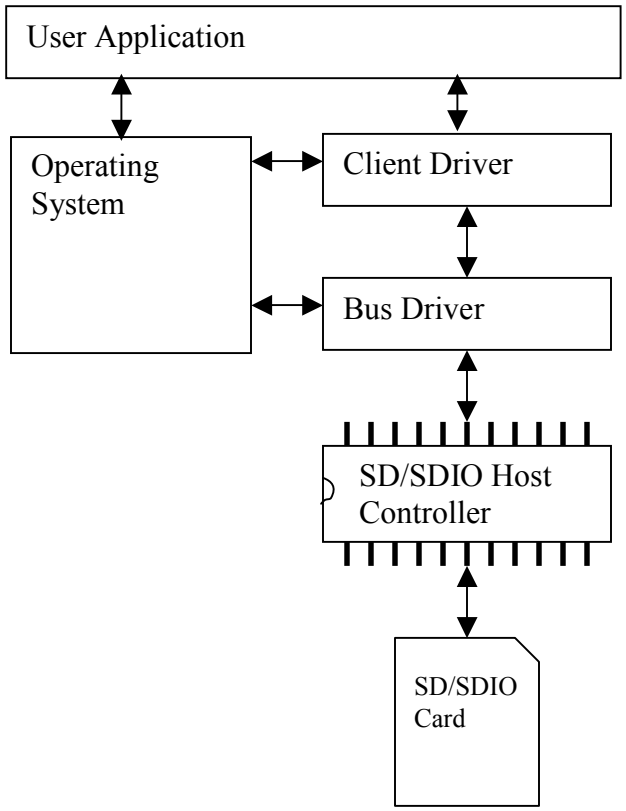
A specific class of cards that implement an SD memory function in conjunction with one or more I/O functions is a combo card. This specific class of device requires special attention to accommodate the fact that the card is actually an SD memory controller embedded along side an SDIO controller function. The intent is to allow the SD memory controller to behave normally in non-SDIO aware hosts with the SDIO controller, including I/O functions, to be completely disabled. In SDIO capable hosts, the plug and play software can load the appropriate control software that enables the SDIO function while still controlling the memory portion as a normal memory card. An example of a combo card is a GPS card containing a memory function[1] to store navigation data.

[1.] SDIO functions can implement at most 2MB of non-volatile storage for code or application specific data in their code storage (CSA) memory space. Using more than 2MB, may require the vendor to license separate patents for flash based storage cards from the original members (the "3C") of the SD consortium.

4.0 SDIO Software Architectures

4.1 Driver Model

Secure Digital I/O system software is a collection of modules that performs common and application specific configuration of I/O functions and normal card operation. A typical system would consist of modules that perform the basic card detection and configuration in one common module while application specific modules perform the card-specific configuration and normal run-time operation. The common module is also referred to as the bus driver. A bus driver handles SD/SDIO-specific card detection and performs enough initialization of the card to retrieve plug and play identifiers (Class code or CIS manufacturer tuple). Using the plug and play information the bus driver may prompt the operating system to load additional application specific modules (known as client or function drivers) to control and operate the card. A bus driver also provides services to these drivers (clients) to manage the flow of SD bus requests/packets. This client-server relationship can be embodied in a software stack, shown in the diagram below:



The bus driver provides a layer of hardware abstraction between the client drivers and the host controller hardware. Client drivers written to this abstracted interface are easier to

port to other systems that may have the same bus driver software interface. It is entirely possible to implement this architecture for systems with or without an operating system. Some deeply embedded systems may actually employ rudimentary task scheduling and the stack can be designed as a single state machine without the support of multithreading or preemptive multitasking.

The choice of supporting SD native or SPI bus modes may be restricted by the choice of hardware. Many off the shelf microcontrollers utilize on-board SPI controllers and cost sensitivity may require that the software stack support an SPI approach. In situations where an embedded system has a single purpose and can dedicate enough CPU cycles, a SPI approach may be more than adequate. SD native support requires special controllers and is often only available on higher cost, more full featured system-on-chip devices.

4.2 Software Performance

The expected performance of this architecture can be constrained by the system's processing power, controller hardware design and operating system (or a non-OS execution environment). The number of transition layers between and within modules affects the speed and CPU utilization of requests flowing down the stack. Given the increase in compute power, this overhead is decreasing and often almost negligible. If the bus driver manages requests in a multithreaded environment, synchronization primitives such as locks, mutexes, and semaphores may add additional overhead. In the multi-threaded environment the bus driver acts as a gatekeeper to prevent multiple client drivers from accessing the host controller at the same time. Multithreading does make complex code more manageable and often increases the overall data throughput but may not be the best method in all situations, especially in resource-constrained systems.

Since most host controllers do not directly map the SDIO card register space into the processor local space, updating a single 8-bit value is a non-trivial task. Most host controllers are merely serial interface engines that implement the basic shuttling of commands, responses and data to and from cards. Host controllers are usually flexible enough such that new commands can easily be added to the specification without much change. This leaves the task of specifying the commands and the modes of operation to software. For example, to update an 8 bit register in SDIO space using CMD52, software must specify to the underlying hardware that CMD52 is to be sent with an argument field consisting of the register address, a read/write bit and an optional value (write operation only). In software this is represented as an SD bus request structure or object. SD bus requests have some overhead in that they must be managed appropriately for a given architecture. They contain information on the SD bus transfer (commands, parameters, flags, buffers) and may reference operating system specific objects. There is usually some overhead in allocating and processing SD bus requests. Some of this is attributed to operating system characteristics such as synchronization and context switching overhead. Good bus driver designs mitigate the overhead by efficiently managing and reusing objects while minimizing required locking. There may be interrupts and context switches involved when SD bus requests are completed, typically followed by the

notification of waiting software modules. The trivial read-modify-write register operations, which may have taken a few CPU cycles, can now result in thousands of cycles.

As with all new standards, some specifications are not met in the initial design of cards and controllers. Some flexibility is often necessary in software to accommodate designs that may exclude features or do not conform exactly to the intended specification. It is a good practice to provide some methods that card control software can employ, such as altering the card's mode of operation (i.e. initialization, clock rate, bus width), to work around silicon issues.

5.0 SDIO Host Performance

5.1 Electrical Interface

It is tempting to brush aside the fact that SD uses a 25 Mhz clock rate and not take signal integrity seriously. A relatively slow clock compared to the 50 Mhz local buses and 100 Mhz SDRAM interfaces reduces SD signal integrity to a low priority. More and more silicon implementations are being done in high-speed CMOS processes with high edge rate I/O pins at any frequency. Fast rise and fall times can induce detrimental line reflection for a given PCB trace length and board characteristics. A high edge rate on a relatively long trace length with many discontinuities (vias, corners, etc.) can cause excessive ringing and overshoot. SD controllers should implement slew rate controlled I/O to ramp the edge rates. Slew rate control can eliminate line reflection and still meet the 25 Mhz timing budget. The SDIO specification does not specify active or passive termination and the best option is to keep trace lengths from the controller to the slot as short as possible and minimize discontinuities.

5.2 Clocking Issues

All SD/SDIO transactions are accompanied by a clock signal for synchronizing the command, response and data frames. The host drives the clock and may remove the clock source at any time. There is one exception to this rule and that is for SDIO cards operating in 4-bit mode with interrupts enabled. Section 3.2 provides a discussion of this clocking requirement. A host controller should have the capability to turn off the clock after each transaction. This will reduce power consumption and some device datasheets indicate that it can reduce active current on some cards by as much as 8-20mA, a significant amount of current for a battery powered device. For 1-bit SDIO cards and memory-only cards, turning off the clock is effective and painless. A 4-bit SDIO card that uses an interrupt however must have the clock applied continuously for the detection of its interrupts. Unless control software can actively manage the bus mode switching, the clock should remain on at the consequence of drawing more power. A client driver could determine, in an application specific way, that the device will not expect data for some time (idle detection) and can drop the bus mode to 1-bit to allow the removal of the SD clock.

The SD clock rate should range from 100Khz (typical initialization clock rate) to as high as 25 Mhz. 400 KHz should also be supported for SDIO low speed devices. The lower clock rates will reduce power consumption in the interface and the general rule is to never over-clock a card interface but to use the next lower clock rate to insure that the card can operate reliably. It is also recommended that the intermediate frequency of 20 Mhz be supported to fully utilize the MMC interface.

5.3 Power Management

Voltage requirements of SD cards can vary from manufacturer to manufacturer and card to card. Virtually all cards minimally support the 3.0-3.6V range and most host systems supply 3.3Vnominally. To gain additional power savings, some SD memory cards can be powered down to as low as 1.8V, further reducing both active and standby power. At this voltage some cards may disable their power wasting linear regulators and operate at a direct voltage for maximum efficiency. The SD specification allows software to query a cards voltage capability during card identification (typically at 3.3V) and then reduce voltage to a more optimal setting. Flash write performance is usually not affected by the lower voltage as internal charge pumps should supply the necessary voltage and current for erasure. Consultation with a card manufacturer may offer better insight into how to manage power consumption.

Power is addressed in the SDIO standard to prevent over-current situations on legacy SDIO capable systems. The 1.1 version of the standard introduces a power capabilities tuple in the CIS that can aid in determining whether a system should fully power a card. Additional registers are defined to enable high-powered modes that only a 1.1 compliant SDIO system would be capable of activating. Given a fixed power budget, a system could selectively operate a card at lower performance to maximize battery life.

Reducing card power during normal operation is application specific. This requires that the client drivers monitor (or are notified of) system power states and are given the opportunity to act accordingly. Most embedded operating systems in battery-powered environments provide some methods to determine the current system power state. The card's driver can then perform the necessary control actions to bring the card into an appropriate power state. Additionally a driver may keep an SDIO card disabled or operating at very low power until a user application opens a handle to the device instance.

5.4 Controller Performance

An SDIO host controller transforms software SD command requests into an SD or SPI bit stream to the target card. The host controller issues start and stop bits (framing), drives the clock and in most implementations (except SPI) a hardware generated CRC code follows commands and/or data. The host controller also buffers incoming responses and/or data frames from target cards. Aside from the SD clock rate, host performance can be affected by the modes in which data is transferred (Programmed I/O or DMA) and the number of interrupts required per transaction.

Programmed I/O:

Programmed I/O requires bus data to flow through the processor onto the controller through a set of registers (i.e. a Data port). A data port may be a FIFO allowing the processor to rapidly load data and wait for the controller to complete the transfer. The size of the FIFO can greatly impact programmed I/O performance since it effectively

determines the number of interrupts for each transaction. A FIFO empty or full interrupt requires attention from the processor's interrupt service routine to reload or drain the FIFOs. A system should minimally support a 512-byte FIFO size for optimal SD card performance, which translates into 1 interrupt per block in a multi-block data transfer. The consequence of programmed I/O is more apparent when trying to fully utilize the available SD bus bandwidth. Each time a FIFO becomes full or empty the SD bus is effectively paused until the interrupt routine can reload or drain the FIFO. In most designs as soon as the FIFO is filled with at least 1 byte, the transfer can continue immediately. Increasingly faster processors and the use of real time OSes are reducing/limiting interrupt latency times making this overhead negligible in comparison to the SD transfer time.

 Larger FIFOs are certainly better and SDIO transfer sizes can be up to 2048 bytes per block, however 512 bytes has been adequate for most applications in practice.

Direct Memory Access:

Direct memory access can reduce the need for internal FIFO buffers as system memory can be used directly in the transfer. This can greatly reduce the number of interrupts per transfer as the DMA controller can transfer the entire data payload autonomously. DMA can be implemented using a common buffer approach, essentially a fixed size contiguous block of memory for DMA use, or through a scatter-gather approach using scattered memory pages mapped into a virtual memory system. The later requires the fewest number of interrupts but is the most complex to implement in terms of software and hardware. DMA controllers are complex and many system-on-chip designs are limited in there use of DMA. A host controller can implement its own bus mastering DMA (i.e. PCI Local Bus) or require a discrete multi-channel DMA controller to shuttle data to and from main memory to an internal or external bus (internal SOC bus).

Controller Interrupts

In some situations the interrupt context switch is costly for some simple SD command operations. For example, CMD52 requires approximately 96-100 clocks for the operation. At 25 Mhz, this is an approximate completion time of 4 microseconds. The interrupt switching latency (delay) in some systems can exceed this value and it may make sense to simply poll for completion instead. A polling technique is only usable for command-only requests (no data) as this class of transfer must occur within a relatively short, bounded period of time. Polling completion can also reduce processing overhead by removing the need for asynchronous callback mechanisms or operating system signals.

 For data transfers that may have long write-complete acknowledgement delays, it is not recommended that polling be used. Using interrupts in data transfers are effective if the payload is sufficiently large enough to mask the overhead of the interrupt.

Card Compatibility

A significant step toward card compatibility in systems is the ability of host controllers to support the entire range of SDIO CMD53 block sizes (1-2048). Early host controllers contain some, often bizarre, restrictions on the block sizes of transfers. This sometimes required work-arounds in system software to match card block size to the host controller capable block size.

Some designs are the adaptation of legacy controllers to support SDIO. They may limit the bandwidth and bus width of the SDIO device and may not support certain bus features. This type of system can still operate cards with a respectable level of functionality. Designing a card to operate optimally in this environment (often pre-SDIO spec. finalization) can be challenging, but not impossible to work through.

5.5 System Software

The wide variation in host controller design makes system software implementation very challenging. Selecting an architecture that abstracts the hardware interface to the host controller will maximize code reuse, improve testing and quality of systems. Much of the card configuration and management intelligence can reside in a single common module fully isolated from hardware-specific host controller control code. A new generation of host controllers will often require only a small fraction of the system software stack to be rewritten. Of some importance to deeply embedded systems is the support of SD in different bus modes due to system cost issues. A stack that can be adapted to SPI bus support would be attractive for cost sensitive embedded applications that may not require the speed or flexibility of a full SD native implementation. An architecture that can easily facilitate the inclusion or development of card drivers is also beneficial. This can consist of pre-validated class drivers (i.e. Memory, Bluetooth, GPS) and a framework for creating custom drivers (i.e. APIs, code development tools).

An SDIO technical committee has created a standard host controller specification that defines the register set and behavior for discrete or on-chip SDIO controllers. This allows software developers to maintain one set of re-usable control code for different chip vendors or chip generations. The standard controller also encourages compatibility between systems. The standard host controller defines a register set usable for controllers located on busses such as generic SRAM-like interfaces or PCI local buses. The specification also describes Direct Memory Access techniques, voltage application and card slot electro-mechanical interfaces (card insertion and write protect switch).

6.0 SDIO Card Performance

6.1 Electrical Interface

SD and MMC cards typically operate at their maximum clock rates, however the specification also allows for rates below this for reduced power consumption or imposed limitations on a host controller design. Some host controller designs are limited to a maximum clock rate of 24 Mhz due to the use of widely available 48 Mhz clock sources (i.e. USB clock generators). These designs also use simple clock divider schemes, which consequently can limit MMC operational clock rates to 12-16Mhz.

With respect to SDIO cards, the maximum clock rates are divided into two classifications, normal and low speed cards. Normal cards support a maximum of 25 Mhz while low speed cards have a maximum clock rate of 400Khz. A low speed card typically operates in 1-bit bus modes but can operate in a faster 4-bit mode.

The designer should be aware that the SD clock is merely a transport clock to drive the SD interface and may be removed at any time. Cards should not use the clock to drive application side logic.

There is glaring misconception on the part of SDIO's register addressing model. It would appear that this model provides a simple parallel to serial and serial to parallel bus interface and suggests a low overhead register access bus. This is far from reality. The clock rate is much too low to implement a true serial-to-parallel I/O interconnect like Serial ATA or PCIExpress. SDIO is inadequate for register-based interfaces where performance is necessary and register access is significant in order to maintain the data flow. A host controller exposes a packet interface engine complete with control registers and data port FIFOs and not an SDIO register-space mapping interface. For example, in a memory mapped architecture like PCMCIA, an 8-bit register read/write takes about 0.100-0.300 μs, meanwhile the same operation on SDIO can take up to 4.3 μs at 25 MHz (more than 10 times as long). For a low speed 400 KHz interface, the same 8-bit transfer takes 270 μs. This is just the physical layer interface and does not account for software overhead (interrupt processing, traversing a software stack). Cards like GPS (at low baud rates) can utilize register level access fairly well. The GPS device class is essentially a standard 16550 UART whose registers are exposed through an SDIO interface. The UART has the same programming model as if the UART was locally mapped onto a parallel I/O bus. Instead of simple I/O reads and writes, system software must issue SD commands to read/write registers and even the simplest register access can involve hundreds of CPU instructions. To complicate matters, the 16550 design uses a register and interrupt intensive interface. Typically the FIFOs are small (16 bytes) and line status must be checked for every byte received. At higher baud rates, keeping up with the

UART transceiver becomes a very processor intensive operation and data loss cannot be avoided without some form of flow control (RTS/CTS or XON/XOFF).

6.3 Interrupt Latency

The SDIO bus can assert interrupts in much the same way that traditional parallel bus architectures do. The interrupt grabs the attention of the system processor to attend to an event that has occurred on an I/O function. In many host controller designs this interrupt is just one of many interrupt sources in the controller (i.e. card detect, command complete, FIFO empty). The interrupt does not appear to the system as a separate interrupt level, but instead requires that the host controller software notify some other software module (driver) that an interrupt occurred. The processing of that interrupt may occur in a non-interrupt context to prevent blocking system interrupts for an excessively long period of time. To this respect SDIO interrupt routines are not ISRs in the traditional sense.

SDIO allows for multifunction devices where the single interrupt line is shared among all functions. SDIO interrupts are level sensitive and requires the clearing (acknowledging) of the interrupt source before the interrupt can be re-enabled. The identification of the interrupting function(s) is made via an interrupt status register. The use of the interrupt status register eliminates the need for system software to query every function driver and potentially generating unnecessary bus traffic.

The serial message-based nature of SDIO will increase the latency for SDIO interrupt processing. In order to process interrupts, system software must read one status register and call into the respective driver which may read additional registers such as a local interrupt status register. The driver must also clear or acknowledge the interrupt before returning from it's interrupt handler. Operations in an interrupt handler should be as short as reasonably possible. Further processing may be deferred to a lower priority task thus allowing other I/O functions to handle their interrupts.

6.4 I/O Transfers

A card design can best utilize the SD bus bandwidth by reducing the number of CMD52 register accesses required before data transfers (CMD53) are issued. A flow-through architecture would be ideal where CMD52 accesses are only used at initialization, infrequent configuration, and minimally used in interrupt routines.


A card design that utilizes only CMD52 operations or a mixture of CMD52 and very small CMD53 transfers will naturally have more protocol and processing overhead per data payload bit transferred. This situation should be avoided unless the data transfer rates is significantly smaller that the SD bus and the overhead is deemed negligible.

The majority of SD requests could then be data orientated rather than register setup overhead. Non-block (byte mode operation) or multi-block data commands will improve bus utilization by reducing the effect of the command/response exchange delay inherent in all commands. This is contingent on the data payload size being significantly larger than the command/response. An example of an application that can utilize a flow through architecture is Ethernet or wireless Ethernet. Ethernet configuration is very infrequent and after initialization, packets can flow directly from the system through the card. With network packets of up to 1400 bytes, a single block approach (up to 2048 bytes) or a multi-block approach (less on-chip buffers) would be adequate.

The design decision to use single block (or byte mode) or multi-block mode data transfers is primarily based on application requirements, cost and complexity. SDIO defines CMD53 in two modes of operation, non-block (byte mode) and block mode. Non-block mode or byte mode is just a single block with a limited block payload that can be varied at any time. In byte mode, a single block can be issued with 1-512 bytes at any time. In block mode a system can send a single or multiple blocks with a size of 1-2048 bytes per block. The later mode however requires the programming of the I/O Block size register with the expected bytes per block. The block size is then fixed for each block for the entire duration of the transfer. In systems where partial blocks may be issued the transfer is broken down into two transfers, one with blocks of the same length and a final block with the remaining bytes. In between these two transactions the I/O Block size register must be reprogrammed with the remaining size. Data padding in the remaining block, up to the block size, can eliminate the need for separate transactions and reduce overhead.

One question that comes up often is: what is the difference between a single block 2048-byte transfer versus sending a multi-block request consisting of 4 blocks of 512 bytes each? The differences arise from a combination of the host controller and the card controller design. From a software perspective there is little difference since they both consist of issuing one CMD53 and transferring the same amount of data (2048 bytes). On the card controller side, the multi-block approach can reduce the card's buffering requirement since each block (512 bytes) requires a write complete acknowledgement. The acknowledgement can delay the host from sending the next block until the card is ready. A single block of 2048 bytes would require the card to minimally buffer the entire 2048 bytes since the card cannot signal the host to "back-off" until the write acknowledgement phase occurs at the end of the block transfer. Most host systems have 500 to 1000 milliseconds timeout delays for write acknowledgement. If a card can process and free buffers within that amount of time then using the multi-block transfer mechanism can reduce card-side buffering requirements. With regard to the host controller, some unforeseen processing overhead may be encountered due to its design. If the host controller FIFO is only 512 bytes, the number of interrupts to load the FIFOs will be roughly the same between the two transfer modes. In some host controller designs the usable FIFO space is limited to the block length and is even imposed on multi-block transfers. This limitation would waste most of the space in a 2048-byte FIFO since it requires that the FIFO be reloaded/drained on the transfer of each block-size (in our 512-byte, multi-block example) worth of data. This limitation is present in some real-world controller designs for the sake of simplicity but is not recommended.

Although DMA is defined in the standard host controller specification, a card designer should not rely on its availability. As application bandwidth requirements increase (closer to 50-100 Mbits/sec) DMA may offer the best and possibly the only solution to achieve the necessary throughput.

7.0 Future of SDIO

SDIO's penetration into the handheld market is increasing with each new generation of consumer devices. Over the next year improvements to the speed (50 Mhz envisioned) and the addition of new device classes will attract more device manufacturers to offer this technology. SD Memory is now the dominant flash storage standard for portable devices and the addition of SDIO comes at very little incremental cost. As SDIO becomes the dominant form factor, more and more manufacturers of Compact I/O cards may migrate to SDIO.

SDIO may evolve as an on-board interconnect for modules such as wireless LAN, digital cameras or GPS sub-assemblies, supplanting traditional board interconnects like I2C or UART. SDIO provides superior bandwidth compared to these interconnects and with the availability of standard device classes, makes integration much simpler. The demand for new technology forces manufacturers to buy and integrate rather than make components in order to reach market faster. SDIO as an interconnect can be the technology to integrate these components together quickly and effectively.

8.0 Conclusion

SDIO as an I/O expansion technology fulfills a gap in the mobile device market. That is the ability for consumers to easily upgrade or add new functionality to their portable devices. Driven by standards, and supported by over 500 companies, the momentum for this technology will continue to grow.

As with all standards, early adopters must wrestle with design problems and tradeoffs due to the lack of available chips and early non-field tested solutions. Design issues that can impact functionality and performance can be encountered in number of areas:

- Host controller silicon design and implementation (includes host controller programming model).
- Host system software design.
- Card controller silicon design and implementation (includes card programming model).
- Card control software.

An understanding of SDIO issues with regard to implementation and performance can better prepare the designer to ask the right questions and make the proper assessments throughout the course of their investigation, design and production.

About the author

Seung Yi is the Vice President and Chief Technologist of Codetelligence, Inc. He has over 12 years of experience developing embedded software ranging from low-level 8-bit motor control applications to high performance 32-bit networked and mobile computing platforms. As a former senior engineer at BlueWater Systems and Bsquare Corporation, Seung Yi was involved in many software projects and products using Windows and Windows Embedded (CE, XP) operating systems. Seung Yi graduated from Kettering University with a B.S. in Electrical Engineering. In his spare time he enjoys spending time outdoors in the summer, snowboarding in the winter and electronic gadgets/toys in between.

Email : seungyi@codetelligence.com
Main : 206-527-4344 ext.7
Web : www.codetelligence.com

Codetelligence, Inc is a software solutions company focusing on innovative knowledge delivery tools for the embedded software market. Codetelligence also provides consulting services on a variety of embedded software topics including technology consulting, architecture design, software design and trouble-shooting.