4D LABS

GOLDELOX-DOS Command Set

Software Interface Specification

Document Date: 2nd April 2009
Document Revision: 1.0

Table of Contents

1. Host Interface

The GOLDELOX-DOS chip is a slave peripheral device and it provides a bidirectional serial interface to a host controller via its UART. All communications between the host and the device occur over this serial interface. The protocol is simple and easy to implement, without the overhead of complicated packet-based protocols.

☞ **Serial Data Format: 8 Bits, No Parity, 1 Stop Bit.**
Serial data is true and not inverted.

1.1 Command Protocol : Flow Control

The GOLDELOX-DOS is a slave device and all communication and events must be initiated by the host. Each command is made up of a sequence of data bytes. When a command is sent to the device and the operation is completed, it will always return a response. For a command that has no specific response the device will send back a single acknowledge byte called the ACK (06hex), in the case of success, or NAK (15hex), in the case of failure.

Commands having specific responses may send back varying numbers of bytes, depending upon the command and response. It will take the device a certain amount of time to respond, depending on the command type and the operation that has to be performed. If the GOLDELOX-DOS chip receives a command that it does not understand it will reply back with a negative acknowledge called the NAK (15hex). Since a command is only identified by its 'position' in the sequence of data bytes sending incorrect data can result in wildly incorrect operation.

1.2 Serial Setup : Auto-Baud

The GOLDELOX-DOS has an auto-baud feature which can automatically detect the host speed and can set its internal baud rate to operate from 300 to 256K baud. Prior to any commands being sent to the module, it must first be initialized by sending the auto-baud character '**U**' (55hex) after any power-up or reset. This will allow the module to determine and lock on to the baud rate of the host automatically without needing any further setup. Once the device has locked onto the host baud rate it wil respond with an ACK byte (06hex).

☞ **Auto-Bauding must be performed each time the device is powered up or reset.**

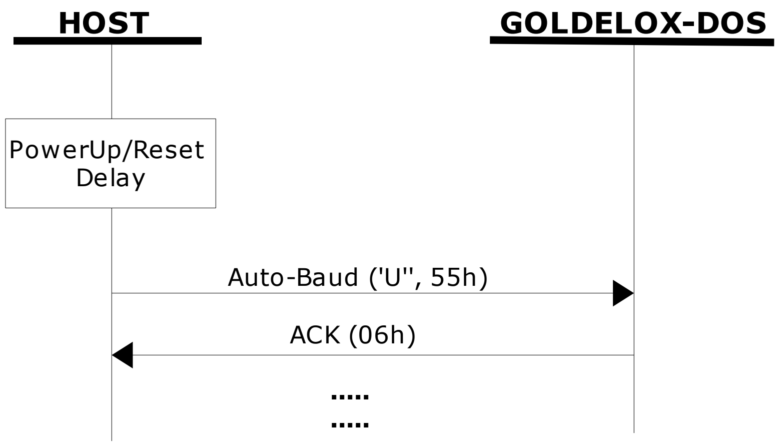
If the host needs to change the baud rate, the GOLDELOX-DOS must be power/reset cycled. The "U" command cannot be used to change the baud rate during the middle of normal usage.

1.3 Powerup and Reset

When the GOLDELOX-DOS device comes out of a power up or external reset, a sequence of events must be observed before attempting to communicate with the module:

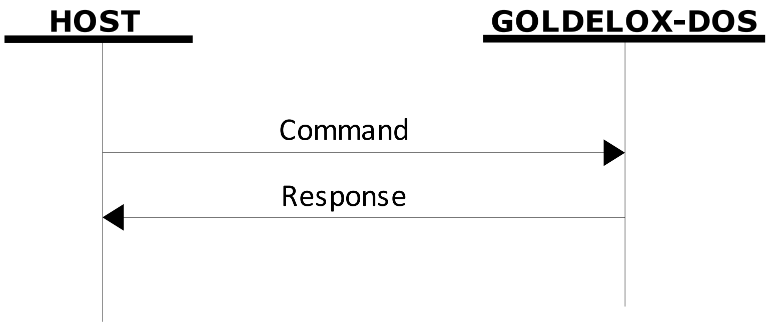
- Allow up to 500ms delay after power-up or reset for the GOLDELOX-DOS to settle. Do not attempt to communicate with the device during this period. The device may send garbage on its TX Data line during this period, the host should disable its Rx Data reception.

- The host transmits the auto-baud character (capital **U**, **55**hex) as the first command so the device can lock onto the host's baud rate.
- Once the host receives the ACK, the GOLDELOX-DOS is now ready to accept Disk Drive commands from the host.



2. Command Set

The data storage and retrieval takes place via the serial interface. A handful of easy to learn commands provide complete access to the memory card for data reads and writes. The simplified command set also means that very low overheads are imposed on the host controller. Commands and responses can be either single bytes or many bytes. All commands return a response, either an acknowledge or data.



The command set is grouped into 3 sections:

- General Commands
- Low-Level (RAW) Disk Drive Commands
- FAT-Level Disk Drive Commands

Each Command set is described in detail in the following sections.

☞ **Seperation characters such as commas ',' or spaces ' ' or brackets'(' ')' between bytes that are shown in the command/response syntax descriptors are purely for legibility purposes and must not be considered as part of any transmitted/received data unless specifically stated.**

2.1 General Commands

2.1.1 Autobaud

Command	**cmd**	
	cmd	**55**(hex) or **U**(ascii) : Command header byte
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte
Description	This must be the very first command sent to the GOLDELOX-DOS after powerup or reset. This will enable the device to lock on to the host baud rate.	

2.1.2 Version/Device Info Request

Command	**cmd**	
	cmd	**56**(hex) or **V**(ascii) : Command header byte
Response	**device_type, silicon_rev, pmmc_rev, reserved1, reserved2**	

	device_type	**03**(hex) : This byte indicates the device type is GOLDELOX-DOS
	silicon_rev	This byte indicates the GOLDELOX silicon revision
	pmmc_rev	This byte indicates the PmmC firmware revision
	reserved1	This byte is reserved for future support. If the value is 0 then ignore it
	reserved2	This byte is reserved for future support. If the value is 0 then ignore it
Description	This command requests all the necessary information from the device about its characteristics and capability.	

2.1.3 FAT Protect

Command	**cmd, mode, value**	
	cmd	**59**(hex) or **Y**(ascii) : Command header byte
	mode	**08**(hex) : FAT protection byte
	value	**00**(hex) : Protection = OFF **01**(hex) : Protection = ON
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if sucessful **15**(hex) : NAK byte if unsucessful or card not present
Description:	This command protects the FAT file system (if present) on the card from being read or written to by low level (RAW) commands. If the memory card contains any FAT (FAT16 or FAT32) partition, when the initialize command is executed or the device comes out of a reset, FAT Protection is turned ON automatically. This means the host will not be able to access the card using Low-Level (RAW) read or write commands unless it subsequently turns off the FAT protection. For a 'Non Standard' card containing two partitions, one FAT and one RAW (Partition type **DA**hex), the default is ON. In this case FAT reads and writes will occur to the FAT partition and RAW reads and writes will be offset into the RAW partition. i.e. a write to sector 0 will write to sector 0 in the raw partition. FAT32 is currently not supported. If you mount a FAT32 formatted disk, you will not be able to access it at all, both FAT and RAW commands will fail. You can either reformat the memory card as FAT or unprotect the card and replace sector 0 with 512 hex 00s.	

2.2 Low-Level (RAW) Disk Drive Commands

The following commands are related to Low-Level Disk Drive operations and they are described in this section. If the memory card contains any FAT (FAT16 or FAT32) partition, when the initialize command is executed or the device comes out of a reset, FAT Protection is turned ON automatically. This means the host will not be able to access the card using Low-Level (RAW) read or write commands unless it subsequently turns off the FAT protection. Use the "FAT Protect" command described in section 2.1.3.

2.2.1 Initialise Disk Drive Memory Card

Command	**ext_cmd, cmd**	
	ext_cmd	**40**(hex) or **@**(ascii) : Extended Command header byte
	cmd	**69**(hex) or **i**(ascii) : Command header byte
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if sucessful **15**(hex) : NAK byte if unsucessful or card not present or if there is no RAW partition and the FAT protection is turned ON.
Description	This command initialises the memory card. The memory card is always initialised upon Power-Up or Reset cycle, if the card is present. If the card is inserted after the power up or a reset then this command must be used to initialise the card. If there is any FAT (FAT16 or FAT32) partition on the card, using this command will automatically protect the card and turn FAT Protection ON. This is implemented for safety reasons so that any important information in the card is not overwritten. The GOLDELOX-DOS device will not respond to any of the Low-Level (RAW) commands in this section until FAT protection is turned OFF. Use the “FAT Protect” command described in section 2.1.3. **Note!** There is no card insert/remove auto detect facility. This command is duplicated also in section 2.3. It is the same command, there is no difference.	

2.2.2 Read Sector Block Data

Command	**ext_cmd, cmd, SectorAdd**(hi:mid:lo)	
	ext_cmd	**40**(hex) or **@**(ascii) : Extended Command header byte
	cmd	**52**(hex) or **R**(ascii) : Command header byte
	SectorAdd	A 3 byte sector address (big endian). Sector Address range from 0 to 16,777,215 depending on the capacity of the card. Each sector is 512 bytes in size. There are 2048 sectors per every 1Mb of card memory.
Response	**data(1..512)**	
	data	512 bytes of sector data
Description	This command will return 512 bytes of data relating to a sector.	

2.2.3 Write Sector Block Data

Command	**ext_cmd, cmd, SectorAdd**(hi:mid:lo), data(1..512)	
	ext_cmd	**40**(hex) or **@**(ascii) : Extended Command header byte
	cmd	**57**(hex) or **W**(ascii) : Command header byte
	SectorAdd	A 3 byte sector address (big endian)
	data	512 bytes of sector data. Data length must be 512 bytes.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if sucessful **15**(hex) : NAK byte if unsucessful or card not present or if there is no RAW partition and the FAT protection is turned ON.

Description	This command allows downloading and writing blocks of sector data to the card. The data block must always be 512 bytes in length. For large volumes of data such as images, the data must be broken up into multiple sectors (chunks of 512 bytes) and this command then maybe used many times until all of the data is written. If the data block to be written is less than 512 bytes in length, then make sure the rest of the remaining data are padded with 00hex or FFhex (it can be anything). If only few bytes of data are to be written then the **Write Byte** command can be used. Once this command is sent, the device will take a few milliseconds to write the data into its memory card and at the end of which it will respond. Only data(1..512) are written to the sector. Other bytes in the command message do not get written.

2.2.4 Set Memory Address

Command	**ext_cmd, cmd, Address**(Umsb:Ulsb:Lmsb:Llsb)	
	ext_cmd	**40**(hex) or **@**(ascii) : Extended Command header byte
	cmd	**41**(hex) or **A**(ascii) : Command header byte
	Address	A 4 byte card memory address (big endian) for byte wise access.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if sucessful **15**(hex) : NAK byte if unsucessful or card not present or if there is no RAW partition and the FAT protection is turned ON.
Description	This command sets the internal memory address pointer for byte wise reads and writes. After a byte read or write, the memory Address pointer is automatically incremented internally to the next byte address location.	

2.2.5 Read Byte Data

Command	**ext_cmd, cmd**	
	ext_cmd	**40**(hex) or **@**(ascii) : Extended Command header byte
	cmd	**72**(hex) or **r**(ascii) : Command header byte
Response	**data_byte**	
	data_byte	1 byte of card data
Description	This command provides a means of reading a single byte of data back from the card. Before this command can be used, memory address location must be set using the **Set Memory Address** command. Once this command is sent, the device will return 1 byte of data relating to that memory location set by the memory address pointer. The memory address location pointer is automatically incremented to the next byte address location.	

2.2.6 Write Byte Data

Command	**ext_cmd, cmd, data**	
	ext_cmd	**40**(hex) or **@**(ascii) : Extended Command header byte
	cmd	**77**(hex) or **w**(ascii) : Command header byte

	data	1 byte of card data
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if sucessful **15**(hex) : NAK byte if unsucessful or card not present or if there is no RAW partition and the FAT protection is turned ON.
Description	This command permits writing single bytes of data to the card. This is useful for writing small chunks of data at irregular intervals quickly. For large data blocks it is more efficient to use the **Write Sector Data** command described previously. Before this command can be used, the card memory address location must be set using the **Set Memory Address** command. Once the **Write Byte** command is sent, a single byte of data will be stored to that memory location set by the memory address pointer. The memory address pointer is automatically incremented to the next location. Only the **data** byte is written. Other bytes in the command message are not stored.	

2.3 FAT-Level (DOS) Disk Drive Commands

The following commands are related to FAT-Level Disk Drive operations and they are described in this section. If the memory card contains a FAT (or FAT32) partition when the initialize command is executed, FAT Protection is turned ON automatically. This means the host will not be able to access the card using Low-Level (RAW) read or write commands unless it subsequently turns off the FAT protection.

FAT32 is currently not supported, if you mount a FAT32 formatted disk, you will not be able to access it at all, both FAT and RAW commands will fail. You can either reformat the memory card as FAT or unprotect the card and replace sector 0 with 512 hex 00s.

☞ **FAT16 is referred to as FAT throughout this documentation. At the time of writing, FAT32 is not supported. Future FAT32 and larger capacity (upto 32GB) card support is currently planned.**

2.3.1 Read File

Command	**ext_cmd, cmd, handshaking, "file_name", terminator**	
	ext_cmd	**40**(hex) or **@**(ascii) : Extended Command header byte
	cmd	**61**(hex) or **a**(ascii) : Command header byte
	handshaking	How often the host sends an ACK(06hex) to request more data during transmission:- 00 – No handshaking, Use only for small files (<= 512 bytes) 01 – Once for each byte 02 – Once for each two bytes ... 50(**32**hex), maximum allowed
	file_name	The filename is 1-12 chars long with an assumed '.' between chars 8 and 9, if there is not one specified in the filename.

	terminator	The file_name string must be terminated with a NULL, **00**(hex).
Response	**file_size**(Umsb:Ulsb:Lmsb:Llsb)**, file_data**(1...N)**, acknowledge**	
	file_size	4 bytes of file size (big endian format).
	file_data	Complete file data block : No Handshaking Block of file data : block size determined by value set in handshaking.
	acknowledge	**06**(hex) : ACK byte if sucessful **15**(hex) : NAK byte if unsucessful
Description	Using this command, the host can read a DOS compatible (FAT) file from the memory card. Because the time taken to process the read bytes varies, a technique is required to ensure that the host communications buffer does not overflow and data is not lost. This is implemented by a simple *handshaking* protocol where the GOLDELOX-DOS will break up the file into smaller data blocks. When the host receives a block, it sends an ACK(06hex) to request the next block of data. The size of the data block is initially set by the host in the command packet, specified by the value in the "handshaking" byte. The larger the value the better, as long as the host system can buffer the incoming block size. Setting this value too low will slow the transfer. The first 4 bytes sent by the device, after receiving the command packet, represent the size of the requested file (Umsb:Ulsb:Lmsb:Llsb). The host then responds with an ACK(06hex) to indicate it wants the file, or NAK(15hex) if it wishes to terminate the receive. The first block of data bytes (block size set by the handshaking value) of the file are then sent from the device, the host then responds with another ACK(06hex) to receive the next block of bytes. This process continues until all of the file data has been received. The device responds with a final ACK if the transfer completes successfully, otherwise it responds with a NAK. The final ACK is not part of the handshaking.	

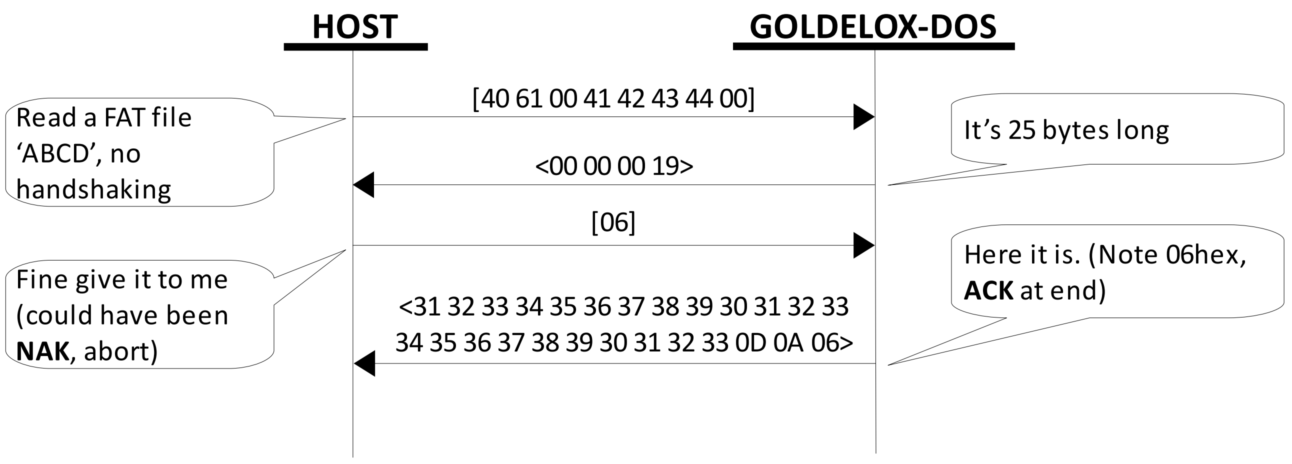

HOST
GOLDELOX-DOS
Read a FAT file 'ABCD', no handshaking
[40 61 00 41 42 43 44 00]
It's 25 bytes long
<00 00 00 19>
[06]
Fine give it to me (could have been NAK, abort)
Here it is. (Note 06hex, ACK at end)
<31 32 33 34 35 36 37 38 39 30 31 32 33 34 35 36 37 38 39 30 31 32 33 0D 0A 06>

Diagram 1: Read File – No Handshaking

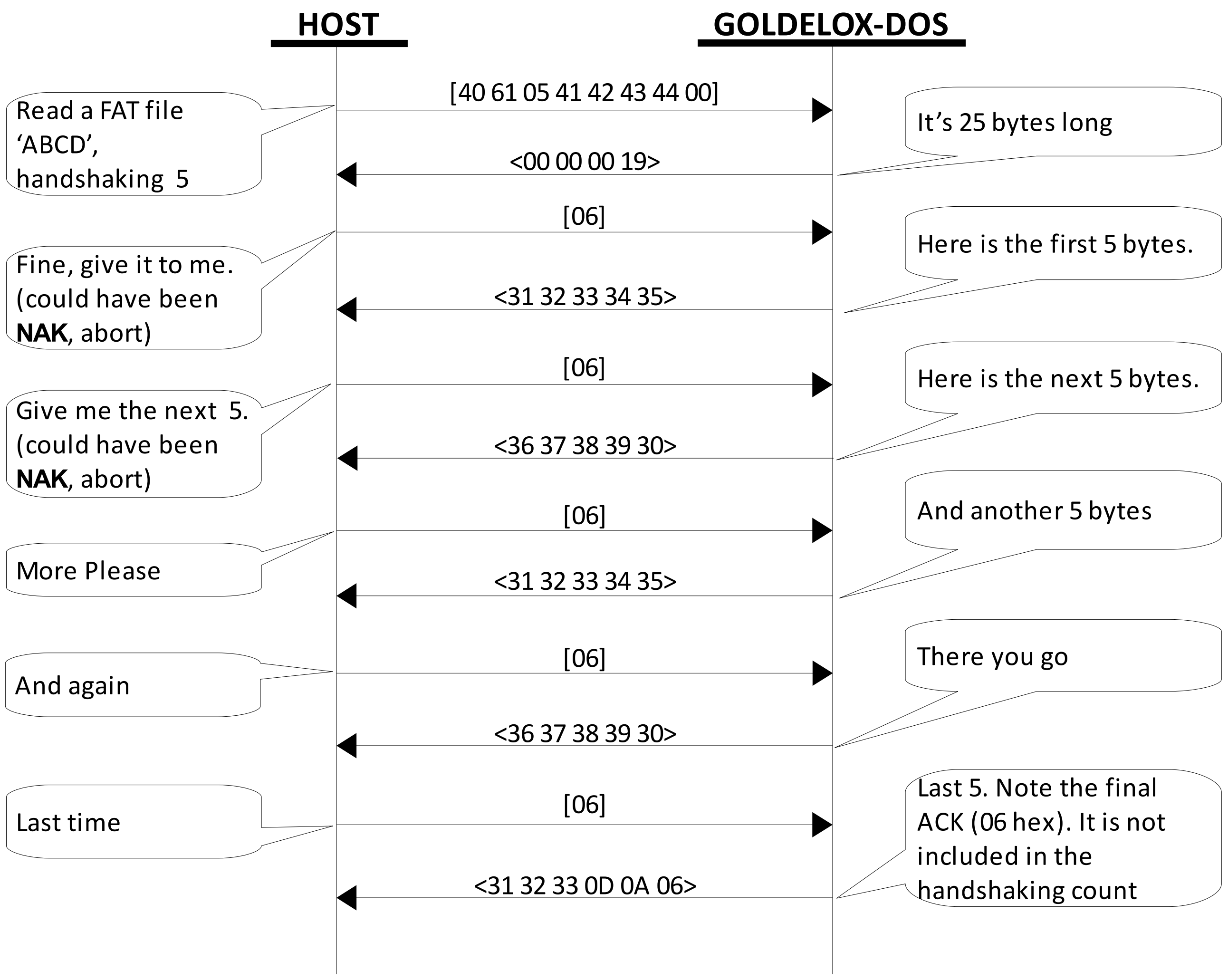

HOST
GOLDELOX-DOS
Read a FAT file 'ABCD', handshaking 5
[40 61 05 41 42 43 44 00]
It's 25 bytes long
<00 00 00 19>
[06]
Fine, give it to me. (could have been NAK, abort)
Here is the first 5 bytes.
<31 32 33 34 35>
[06]
Give me the next 5. (could have been NAK, abort)
Here is the next 5 bytes.
<36 37 38 39 30>
[06]
More Please
And another 5 bytes
<31 32 33 34 35>
[06]
And again
There you go
<36 37 38 39 30>
[06]
Last time
Last 5. Note the final ACK (06 hex). It is not included in the handshaking count
<31 32 33 0D 0A 06>

Diagram 2: Read File - Handshaking other than 0

2.3.2 Write File

Command	**ext_cmd, cmd, options, "file_name", terminator, filesize**(Umsb:Ulsb:Lmsb:Llsb)**, file_data**(1 .. N)	
	ext_cmd	**40**(hex) or **@**(ascii) : Extended Command header byte
	cmd	**74**(hex) or **t**(ascii) : Command header byte
	options	Controls handshaking (how often the device sends an ACK to request more data from the host) and whether an existing file is appended to. **Handshaking:** 00 – No handshaking, limit this to small files (<= 100 bytes) 01 – Once for each byte 02 – Once for each two bytes ... 50(**32**hex), maximum allowed **Append Mode:** **00**(hex) – No Append, file will be created (or overwritten if it exists). **80**(hex) – Append mode, file will be appended to (or created if it doesn't exist). **Note:** The two options are added, or **OR**ed together to produce the final options, eg **82**(hex) would indicate handshaking for every two bytes and Append mode.
	file_name	The filename is 1-12 chars long with an assumed '.' between chars 8 and 9, if there is not one specified in the filename.
	terminator	The file_name string must be terminated with a NULL, **00**(hex).
	file_size	4 bytes of file size (big endian format).
	file_data	Complete file data block : No Handshaking Block of file data : block size determined by value set in handshaking.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if sucessful **15**(hex) : NAK byte if unsucessful
Description	This command allows the host to write a DOS compatible (FAT) file to the memory card. The GOLDELOX-DOS device serial port (UART), has a buffer size of 512 bytes for capturing incoming data from the host. If this buffer fills up and overflows, data will be lost. Therefore the host must allow the device enough time to write its buffer to the memory card so it can then receive further file data from the host. For small files (less than 100 bytes in the append mode) the host can send the complete file data in one attempt. However, for larger files a simple *handshaking* protocol is implemented where the host sends the file data in small blocks. Using this handshaking method, the host always waits for an ACK from the device before sending the next block of data. The size of the data block is initially set by the host in the command packet, specified by the handshaking value in the "options" byte. The larger the value the better. Setting it too low will slow the transfer. The first ACK is always sent by the device, after the filesize parameter is transmitted by the host, or this could also be a NAK in which case one of the parameters is invalid or a file system error occurred. **Note: Do not set handshaking to zero if the file size is larger than 512 bytes.**	

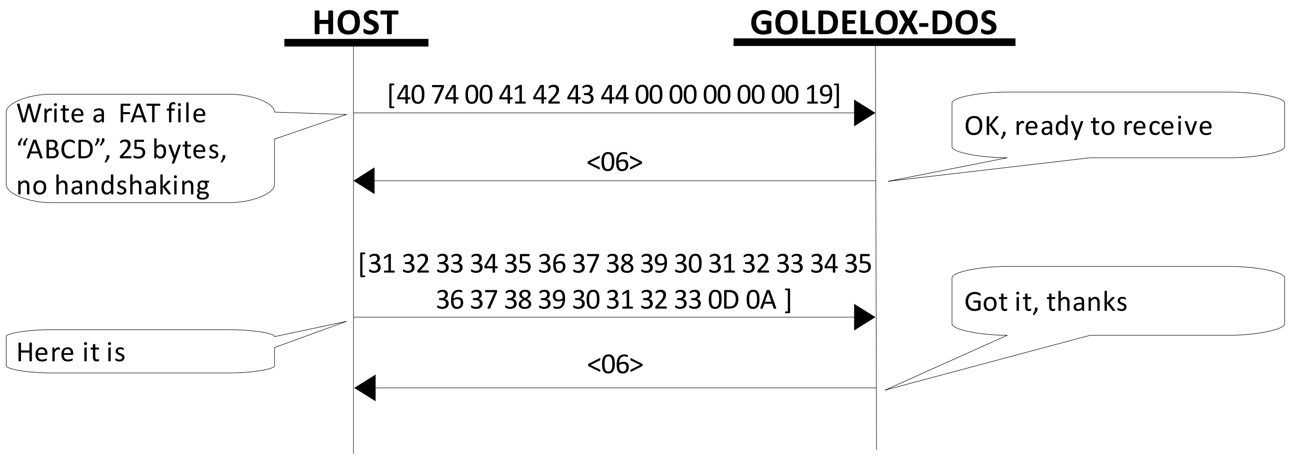

HOST
GOLDELOX-DOS
[40 74 00 41 42 43 44 00 00 00 00 00 19]
Write a FAT file "ABCD", 25 bytes, no handshaking
OK, ready to receive
<06>
[31 32 33 34 35 36 37 38 39 30 31 32 33 34 35 36 37 38 39 30 31 32 33 0D 0A]
Here it is
Got it, thanks
<06>

Diagram 3: Write File - No Handshaking

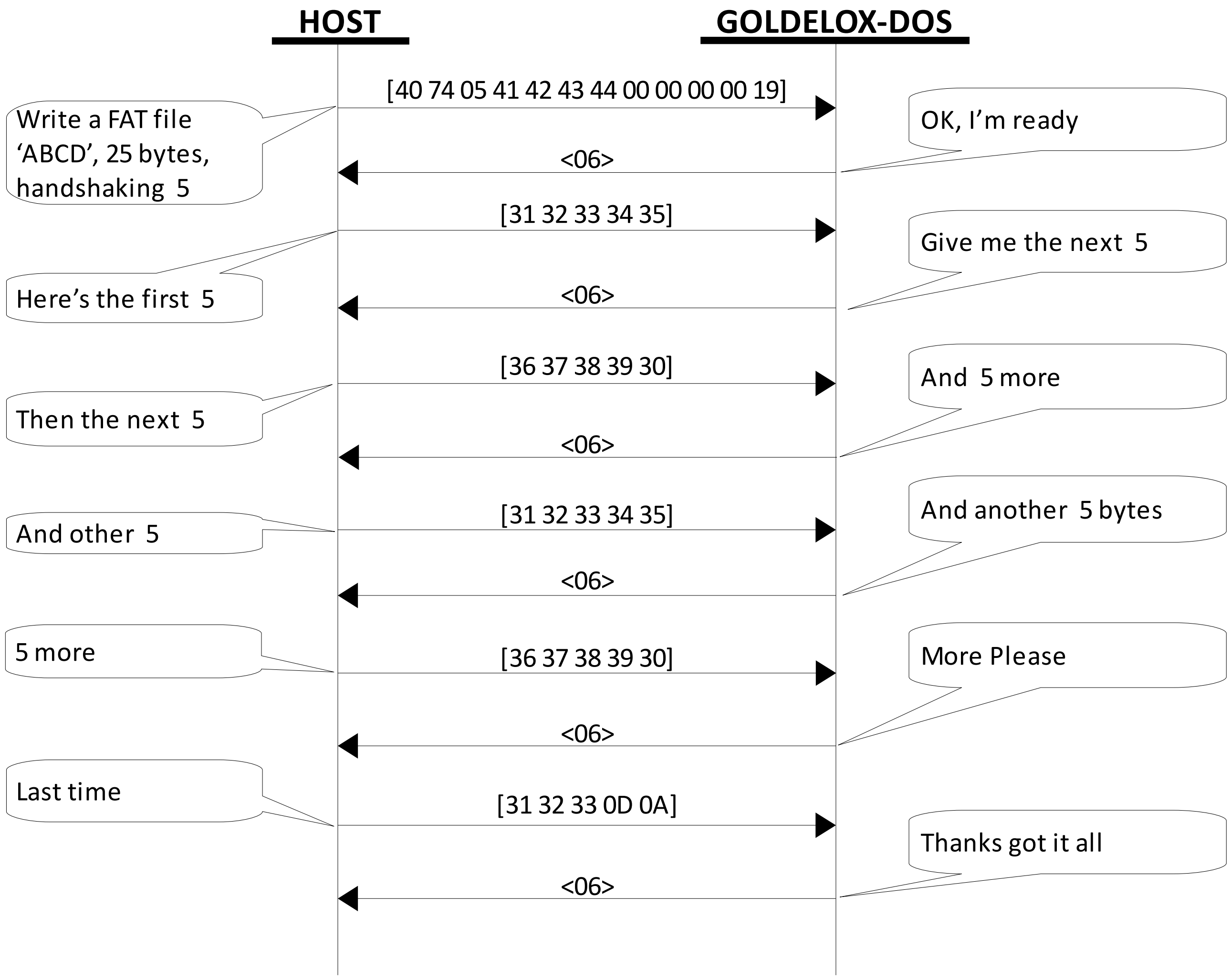


Illustration 4: Write File - Handshaking Other than 0

2.3.3 Erase File

Command	**ext_cmd, cmd, "file_name", terminator**	
	ext_cmd	**40**(hex) or **@**(ascii) : Extended Command header byte
	cmd_hdr	**65**(hex) or **e**(ascii) : Command header byte
	file_name	The filename is 1-12 chars long with an assumed '.' between chars 8 and 9, if there is not one specified the filename.
	terminator	The file_name string must be terminated with a NULL, **00**(hex)
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if file deleted sucessfully **15**(hex) : NAK byte if file not found or an error has occured
Description	Erases the file specified in the "file_name".	

2.3.4 List Directory

Command	**ext_cmd, cmd, "file_name", terminator**	
	ext_cmd	**40**(hex) or **@**(ascii) : Extended Command header byte
	cmd_hdr	**64**(hex) or **d**(ascii) : Command header byte
	file_name	The filename is 1-12 chars long with an assumed '.' between chars 8 and 9, if there is not one specified the filename. Wild cards such as '*' and '?' are allowed.
	terminator	The file_name string must be terminated with a NULL, **00**(hex)
Response	**"fileName1", delimiter, .. , "fileNameN", delimiter, acknowledge**	
	fileName	Character string of the file name in the memory card. Maximum of 12 character bytes (including '.' seperator) are returned in the string for the file name. This will be repeated for all files in the directory. An empty directory with no files or the result of an unsucessful file name or wild card search will only return an ACK. **Note:** At present the GOLDELOX-DOS chip only supports a single directory structure. Future enhancements will support nested directories.
	delimiter	**0A**(hex) : newline
	acknowledge	**06**(hex) : ACK byte if sucessful **15**(hex) : NAK byte if an error has occured
Description	Returns a directory listing (stream of characters) consisting of the files names matching the "file_name" delimited by a Newline(**0A**hex) character. Always responds with an ACK at the completion of a listing. Responds with a NAK if a file error occurs. An empty directory with no files or the result of an unsucessful file name or wild card search will only return an ACK.	

2.3.5 Initialise Disk Drive Memory Card

Command	**ext_cmd, cmd**	
	ext_cmd	**40**(hex) or **@**(ascii) : Extended Command header byte

	cmd	**69**(hex) or **i**(ascii) : Command header byte
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if sucessful **15**(hex) : NAK byte if unsucessful or card not present
Description	This command initialises the memory card. The memory card is always initialised upon Power-Up or Reset cycle, if the card is present. If the card is inserted after the power up or a reset then this command must be used to initialise the card. If there is any FAT (FAT16 or FAT32) partition on the card, using this command will automatically protect the card and turn FAT Protection ON. This is implemented for safety reasons so that any important information in the card is not overwritten. The GOLDELOX-DOS device will not respond to any of the Low-Level (RAW) commands in this section until FAT protection is turned OFF. Use the "FAT Protect" command described in section 2.1.3. **Note!** There is no card insert/remove auto detect facility. This command is duplicated also in section 2.2. It is the same command, there is no difference.	

Proprietary Information

Disclaimer of Warranties & Limitation of Liability