**DragonMMC programming interface.**

**For Firmware V1.30 2020-05-10.**

DragonMMC consists of a ROM containing 6809 code which communicates with an AVR microcontroller which handles all of the SD card and clock related operations. The AVR microcontroller has firmware that talks directly via SPI to the SD card, and handles the FAT filesystem on that card removing the need for the 6809 to do this directly.

The 6809 code communicates with the AVR code through a series of I/O ports that are handled by the onboard Xilinx XC95144XL CPLD, the details of these will be outlined below.

Commands are issued by writing their byte code to D\_CMD\_REG status is read back from the same register once the AVR stops being busy (AVR busy bit of D\_STATUS\_REG is clear).

Generally interchange between the 6809 and AVR will use one of three protocols, bulk data transfer in, bulk data transfer out and latched byte out.

With a transfer in command the 6809 will first issue a CMD\_INIT\_WRITE command, send over the data a byte at a time to the D\_WRITE\_DATA\_REGand then issue a command to be carried out on that data e.g. write it to a file, update the clock etc.

Transfer out commands the 6809 will issue the command, get a status byte and if successful, will then issue a CMD\_INIT\_READ command and then retrieve the data one byte at a time from D\_READ\_DATA\_REGe.g. read data from a file.

Latched byte commands have parameters that are single bytes (up to a maximum of 16), which should be written one after another to D\_LATCH\_REG, and then the command issued in the standard way. **NOTE** The latch write pointer is reset to the first byte after a write to the D\_CMD\_REG a good way of ensuring the byte buffer is flushed is to execute a CMD\_NOP.

**NOTE** All transfers to / from the AVR should check the busy and handshake flags to ensure correct data transfer between the two.

**NOTE** for all transfers IN and OUT are relative to the AVR, e.g. Data or Latch In is 6809→AVR, Data Out is AVR→6809.

**Hardware Details.**

The ports that the 6809 firmware uses to communicate are all in the $FF50-$FF5F region, and are detailed below :

**$FF50 D\_CMD\_REG Command / Result**

All commands are written to this port, and status is read back.

**$FF51 D\_LATCH\_REG Latch**

Used by some commands as a single byte parameters, where a bulk data transfer is not needed the latch register has a 16 byte buffer behind it, each byte written to the latch register is put in the next available buffer position. **NOTE** the buffer position is **RESET** to 0 after a write to the command register **EXCEPT** for the CMD\_INIT\_WRITE command. This is done so that latch and data bytes may be passed in any order. A good way to ensure the byte buffer is clear is to send a CMD\_NOP.

**$FF52 D\_READ\_DATA\_REG Read data**

When transferring data from the AVR to the 6809, it is read a byte at a time from this register.

**$FF53 D\_WRITE\_DATA\_REG Write data**

When transferring data from the 6809 to the AVR, it is written a byte at a time to this register.

**$FF54 D\_STATUS\_REG AVR status**

This register contains the status of the AVR, that the 6809 can read, including the various handshake bits.

**Bit Purpose**

0 AVR\_Busy when this bit is set it indicates that the AVR is busy processing a command, further commands / data should not be sent until this bit is clear.

1 DragonW\_AVRR handshake line from 6809→AVR this bit is set when the 6809 writes a command / data to the registers above, it is cleard once the AVR has read the sent data, indicating that it is ready to receive more data (if busy is not set).

2 AVRW\_DragonR handshake bit from AVR→6809 this bit is set when the AVR has written data and it is ready to be read by the 6809. This bit is cleard when the 6809 reads the data.

3 unused

4 unused

5 unused

6 unused

7 unused

**$FF55 D\_PIA\_MAP PIA redirect (write only)**

Writing to this register enables the re-directing of PIA reads at $FF20 to come from the CPLD instead, this allows the AVR to emulate the FSK input from a tape. Bit 0 is used to control this, setting it to 1 enables emulation, setting it to 0 disables it.

**$FF56 D\_RAM\_CTRL RAM / ROM control register**

Controls the paging of the onboard 6809 firmware ROMS and of the onboard mirror RAMS.

**Bit Purpose**

0 Enable RAM in $8000-$FEFF, this disables the internal ROMs in this area.

1 Write protect the RAM between $8000-$FEFF.

2 Enable the routing of the Q line to the CART pin, to allow a loaded cartridge image to auto start.

3 Enable writes to the $8000-$FEFF to go to the Flash ROM instead of the RAM.

4 A14 of the Flash ROM, this is used to control which bank of 16K is paged into the $C000- $FEFF area. As supplied the Dragon code paged in when this bit is 0, and the CoCo code is

paged in when this bit is 1. On reset the firmware checks which machine it is running on and

sets this as needed.

5 When set the interrupt vectors at $FFF0-$FFFF are read from the top of the first bank of 32K of the SRAM. When cleared they are read from the ROM as normal.

6 Enable snapshot button to generate an NMI to the 6809.

7 Write protect for the interrupt area of RAM.

**$FF57 unused.**

Not currently used.

**$FF58 D\_SAMBITS\_MSB**

**$FF59 D\_SAMBITS\_LSB SAM bits**

Because the SAM does not have a connection to the 6809 data bus, it’s programmed state can not normally be determined. The CPLD on the board, mirrors writes to the SAM registers so the they may be read back, via these two locations. $FF58 contains the 8 most significant bits and $FF59 contains the least significant bits. This allows the SAM’s state to be saved and restored e.g. when creating a snapshot.

**$FF5A / $FF5B RAM bank read / write registers**

If enabled in the RAM / ROM control register (and in map mode 0) the top 32K of the 6809 memory map can be replaced by RAM. This is generally used by the CAS file loader to copy the basic and MMC roms to RAM and then patch them to redirect tape operations to read from a .CAS file on the SD card. The RAM socket may take a static RAM chip up to 512K, the bottom 4 bits of these registers control which block of 32K within the SRAM is accessed. Note that even with the RAMS paged out (in the RAM/ROM control register), writes to this area will **STILL** go to the selected RAM bank. The exception to this is if the Flash write bit is set in the RAM / ROM control register to allow updating of the firmware.

**$FF5C to $FF5F**

Currently unused.

**Command Specification.**

The following is a numerically ordered list of the commands that may be sent from the 6809 to the AVR. Along with their symbolic names as defined in the 6809 and AVR firmware definition files. Wherever possible these should be used directly by including them in any project that is directly talking to the firmware.

Note some of these commands will first require data to be sent to the AVR for them to operate on, with the CMD\_INIT\_WRITE, followed by sending the data to the data out register. This will be noted in the following specifications, by marking the command with Data In.

Commands tagged with Data Out, if successful will place data in the AVR’s buffer to be read by the 6809 by issuing a CMD\_INIT\_READ and then reading the data from the read data register.

All commands will return a status byte which can be checked for errors, except where noted.

**Directory commands**

**CMD\_DIR\_OPEN $00 Data In**

Open the specified directory / wildcard and prepare it to be read this can be used to read the entire contents of a directory by specifying no wildcard (\*.\* will be assumed) or to select a group of files e.g. \*.cas

**CMD\_DIR\_READ $01 Data out**

Reads the next directory entry from a previously opened (with CMD\_DIR\_OPEN) directory, if no more entries are available the STATUS\_LAST bit will be set.

The entry is returned :

offset size use

0 n filename, if directory will be surrounded by < and >

n+1 1 zero terminator

n+2 1 file attributes

n+3 4 file size

**CMD\_DIR\_CWD $02 Data In**

Change to the specified directory.

**CMD\_DIR\_GETCWD $03 Data Out**

Return the currently selected directory.

**CMD\_DIR\_MAKE $04 Data In**

Create the specified directory.

**CMD\_DIR\_REMOVE $05 Data In**

Remove the specified directory.

**CMD\_DIR\_SET\_SNAPPATH $06 Data In**

Set the path to be used when creating snapshots via the NMI menu.

**CMD\_DIR\_GET\_SNAPPATH $07 Data Out**

Get the current snapshot path.

**File Commands**

File commands generally operate on a file identified by a file ID. The file id will be from 0 to NO\_FILES (as defined in the AVR firmware, currently set to 6). The file ID will generally be the first thing written to the latch register.

**CMD\_FILE\_CLOSE $10** **Latch in**

Close the current file, latch contains the file id to close.

**CMD\_FILE\_OPEN\_READ $11 Latch In, Data In**

Open the specified file who’s name is in the AVR buffer, and who’s ID is latched for reading

**CMD\_FILE\_OPEN\_IMG $12 Data In**

Open the specified disk image, the disk image number is specified as the first byte of data immediately followed by it’s zero terminated filename.

**CMD\_FILE\_OPEN\_WRITE $13 Latch In, Data In**

Open the specified file who’s name is in the AVR buffer for writing.

**CMD\_FILE\_DELETE $14 Data In**

Delete the specified file who’s name is in the AVR buffer .

**CMD\_FILE\_GETINFO $15 Latch In, Data In**

Get information about the specified ID returned is as as follows :

Offset Size Purpose

0 4 File size

4 4 File starting block number (within SD card)

8 4 Current file pointer

12 1 File attributes

**CMD\_FILE\_OPENAUTOD $16**

Open the Auto start file for the Dragon (AUTOEXEC.DGN)

**CMD\_FILE\_OPENAUTOC $17**

Open the Auto start file for the CoCo (AUTOEXEC.CCO)

**CMD\_FILE\_OPEN\_OVERWRITE $18 Latch In, Data In**

Open the specified file for overwrite, removing any previous file contents.

**CMD\_FILE\_OPEN\_SNAPR $19 Latch In, Data In**

Open the specified file from the configured snapshot directory for reading.

**CMD\_FILE\_OPEN\_SNAPW $1A Latch In, Data In**

Open the specified file from the configured snapshot directory for writing.

**CMD\_FILE\_OPEN\_STREAMR $1B Data In**

Open a file for stream reading, currently does nothing, as is just a placeholder for completeness.

**CMD\_FILE\_OPEN\_STREAMW $1C Data In**

Open the specified file for streaming to this is used by MMIRROR to rapidly copy bytes from the incoming (real) cassette to the SD card. Puts the AVR in an interrupt driven mode where all bytes written to the data port are double buffered and written to the card, whilst in this mode the AVR will ignore any further commands until the snapshot button is pressed to terminate streaming, close the stream file and return to normal mode.

Will always use file id 0.

**CMD\_FILE\_COPY $1D Data In**

Copy a file to a new file both on the SD card. The filenames are placed in the buffer with the zero terminated source at offset 0, and the zero terminated destination, immediately after the zero terminator of the source.

**CMD\_FILE\_RENAME $1E Data In**

Rename a file on the SD card, data buffer should be formatted as for CMD\_FILE\_COPY.

**CMD\_FILE\_OPENCRE\_IMG $1F Data In**

Open a disk image file if it exists, if it does not exist create it first. Buffer should be formatted as for CMD\_FILE\_OPEN\_IMG.

**Data Transfer and File operation Commands.**

**CMD\_INIT\_READ $20**

Move the AVR’s data pointer to the beginning of the buffer and prepare it to send data to the 6809.

**CMD\_INIT\_WRITE $21**

Move the AVR’s data pointer to the beginning of the buffer and prepare it to receive data from the 6809.

**CMD\_READ\_BYTES $22 Latch In, Data Out**

Read specified number of bytes from open file. The number of bytes to read must have been previously written to the latch register.

**CMD\_WRITE\_BYTES $23 Latch In, Data In**

Write specified number of bytes from open file. The number of bytes to write must have been previously written to the latch register. The bytes to write must have been previously loaded into the AVR’s buffer with CMD\_INIT\_WRITE, and then written to the buffer by sending to the Write data register.

**CMD\_REWIND $24**

Return the file pointer to the beginning of the file.

**CMD\_SEEK $25 Data In**

Seek a file to a specified position….not yet implemented.

**CMD\_TELL $26 Data Out**

Report the current filepointer position of a file, not yet implemented.

**Utility commands.**

**CMD\_GET\_STRLEN $30**

Get the length of the zero terminated string in the AVR buffer. Note does **NOT** set status.

**CMD\_EXEC\_PACKET $3F Data In**

Read and write EEPROM…..not used, may be removed.

**DOS Emulation / Virtual floppy commands**

**CMD\_LOAD\_LBA $40 Data In**

Load the logical block address of the required sector, buffer should be preloaded with :

Offset Size Purpose

0 1 Drive number 0..3

1 4 Logical sector number (little endian e.g. lsb first)

**CMD\_GET\_IMG\_STATUS $41 Latch In**

Returns the file attributes of the specified image number, drive number should be written to the latch register, attributes returned immediately. Note does **NOT** return a status.

**CMD\_GET\_IMG\_NAME $42 Data In**

Returns the path names of the specified images, the drive IDs of the first and last drive images to retrieve should be written to the first two bytes of the buffer. On successful completion the buffer will contain as many zero terminated strings as drives requested. If a drive currently has no image mounted then a zero length string will be returned in it’s position.

**CMD\_READ\_IMG\_SEC $43 Data Out**

Read the specified sector from the image, the sector will have been previously specified with either CMD\_LOAD\_LBA or CMD\_IMG\_SEEK and CMD\_LOAD\_HR. On successful completion the sector data will be in the AVR’s buffer and can be read in the normal manner.

**CMD\_WRITE\_IMG\_SEC $44 Data In**

Write the specified sector to the disk image, the disk will have had the write position set with either CMD\_LOAD\_LBA or CMD\_IMG\_SEEK and CMD\_LOAD\_HR. The sector data should then be loaded into the AVR’s buffer before issuing this command.

**CMD\_SER\_IMG\_INFO $45**

Serialize drive info, saves mounted drives to the settings file in the rood directory of the SD card.

**CMD\_VALID\_IMG\_NAMES $46**

Check to see if image names are valid.

**CMD\_IMG\_UNMOUNT $47 Data In**

Close the specified drive ID’s image file flushing any data and invalidate the drive. The drive id should be loaded into the AVR buffer before issuing this command.

**CMD\_IMG\_SEEK $48 Data In**

Seek the virtual disk to the specified disk track, the AVR buffer should be loaded with the drive ID, followed by the track to seek to.

**CMD\_CREATE\_IMG $49 Data In**

Create a new disk image with the specified geometry. The AVR buffer should be filled as follows :

Offset Size Purpose

0 1 Drive ID 0..3

1 2 Track count LSB first

3 1 Head count

4 1 Sectors per track

The disk image file should have been previously associated with the drive by using CMD\_FILE\_OPEN\_IMG

**CMD\_GET\_FDC\_STATUS $4A**

Get the emulated disk system’s status. The status byte is an emulated version of the status byte that would be returned by the WD2797 / WD1793 floppy controller. The status byte is written immediately, a normal status is **NOT** written first.

**CMD\_READ\_NEXT\_IMG\_SEC $4B Data Out**

Increment the LBA offset by 1 and read the next sector into the AVR buffer, currently broken **DO NOT USE**

**CMD\_LOAD\_HR $4C Data In**

Load the specified head and sector (record) number into the virtual disk, the AVR buffer should be setup as follows :

Offset Size Purpose

0 1 Drive ID 0..3

1 1 Head number

2 1 Sector number

**Cassette emulation commands.**

**CMD\_CAS\_FTYPE $50 Data Out**

Attempt to determine the type of the next file on the emulated cassette and return it. On completion the buffer will contain the single byte filetype.

**CMD\_CAS\_EMULATE2 $5E**

**CMD\_CAS\_EMULATE $5F**

Emulate the tape by way of sending FSK bits to the PIA, these are to allow reading of the SD card to flash the flash ROM image with no firmware already present. They should be used as follows, from basic :

POKE &HFF55,1 : REM Turn on PIA replacement

POKE &HFF50,&H5F : REM give command to AVR

CLOADM

EXEC

These commands should not need to be used under normal circumstances.

**More utility commands**

**CMD\_GET\_CARD\_TYPE $80**

Get the type of card MMC / SD / SDHC etc. Type is returned immediately, command does **NOT** return a normal status.

**CMD\_SET\_BUSY $90**

Set the AVR into busy mode. Should only be used for testing.

**CMD\_NOP $91**

No operation, does nothing but clear the current busy status.

**CMD\_SYNC $92**

Synchronizes any pending data to the card.

**Old AtoMMC commands.**

The following are recognized by the DragonMMC AVR firmware but may be removed in future, they where provided for backward compatibility with the AtoMMC from which the AVR’s firmware was derived.

CMD\_GET\_PORT\_DDR $A0

CMD\_SET\_PORT\_DDR $A1

CMD\_READ\_PORT $A2

CMD\_WRITE\_PORT $A3

**Date and Time related commands.**

**CMD\_GET\_DATETIME $C0 Data Out**

Fetch the current data and time from the real time clock, the buffer is filled as follows :

Offset Size Purpose

0 4 Year in ASCII e.g. 2020

5 1 - seperator

6 2 Month in ASCII, left zero padded.

8 1 - seperator

9 2 Day of month ASCII, left zero padded.

11 1 Space seperator

12 2 Hours in ASCII, 24 hour format, left zero padded

14 1 : seperator

15 2 Minutes in ASCII, left zero padded

17 1 : seperator

18 2 Seconds in ASCII, left zero padded.

20 1 Zero terminator

This is basically the time and date in ASCII in the format YYYY-MM-DD hh:mm:ss

**CMD\_SET\_DATETIME $C1 Data In**

Set the real time clock date and time, the buffer should first be filled in in the same format as returned by CMD\_GET\_DATE.

**Firmware information commands**

**CMD\_GET\_FW\_VER $E0 Data Out**

Get the firmware version along with it’s compile time in ASCII, the buffer will return :

Offset Size Purpose

0 1 Bootloader version packed into a byte with bits 7..4 being the Major version and bits 3..0 being the minor version.

1 n n bytes of zero terminated string containing the firmware build date.

**CMD\_GET\_BL\_VER $E1 Data Out**

Get the AVR bootloader version data, output buffer formatted as for CMD\_GET\_FW\_VER above.

**Configuration byte commands**

**CMD\_GET\_CFG\_BYTE $F0**

Get the specified platform’s configuration byte, the required platform ID must first be written to the Latch register.

The valid platforms are :

PLAT\_SYSTEM $00 DragonMMC system

PLAT\_DRAGON $01 Dragon

PLAT\_COCO $02 CoCo

The configuration byte is returned immediately a normal status is **NOT** returned.

**CMD\_SET\_CFG\_BYTE $F1 Latch In**

The Platform ID and config byte are written to the Latch register and then this command should be executed The bytes should be written in the following order :

Platform id, config byte.

**CMD\_SET\_PLATFORM $F2 Latch In**

Tell the AVR what platform the 6809 end is running on either PLAT\_DRAGON or PLAT\_COCO

**CMD\_READ\_AUX $FD**

Return the latched address of the last register written. Unused, may be removed in future.

**CMD\_GET\_HEARTBEAT $FE**

Each time it is called the heartbeat byte is returned, this will be a sequence of $55 and $AA bytes alternating on each call.

**Status codes.**

For most DragonMMC firmware commands STATUS\_COMPLETE set when a command completes.

If an error condition occours, then STATUS\_ERROR should be set, this way we can use the 6809 BITA / BITB instructions to test for error, this also allows us to BMI on error.

Multi-phase commands like getting directory entries return status complete after each entry. On the last entry this is or'd with STATUS\_LAST.

Status / error name Returned code

STATUS\_ERROR $80

STATUS\_COMPLETE $40

ERROR\_MASK $7F

To be or'd with STATUS\_COMPLETE

STATUS\_LAST $01

To be or'd with STATUS\_ERROR since STATUS\_ERROR can also return a fatFS error code we must make sure these are in a non overlapping range.

ERROR\_INVALID\_CMD $20

ERROR\_INVALID\_IMAGE $21

ERROR\_NO\_DATA $22

ERROR\_INVALID\_DRIVE $23

ERROR\_READ\_ONLY $24

ERROR\_ALREADY\_MOUNT $25

ERROR\_INVALID\_TIME $26

ERROR\_INVALID\_FID $27

**Configuration byte bitmasks.**

System platform (0)

CFG\_ENABLE\_BOOTLOAD $80

Enable bootloader to check for updated AVR firmware on next AVR restart, if found updated firmware will be flashed.

CFG\_DEBUG\_LOG\_REPLY $08

Unknown, unused!

CFG\_BACKUP\_FILE $04

Should we maintain .BAK files when opening files for overwrite that already exist.

CFG\_HIDE\_MAC $02

Should we hide the special files that Apple Mackintosh’s finder creates.

CFG\_DEBUG\_LOG $01

Enable verbose AVR firmware debug messages to it’s TTL serial port, this should normally be off, but may be turned on if developing the firmware, not turning this on will noticeably slow down some operations.

Dragon (1) or CoCo (2) platforms.

CFG\_ENABLE\_AUTOBOOT $40

Should the autostart file be searched for at power on or reset time. This will be AUTOEXEC.DGN for the Dragon or AUTOEXEC.CCO for the CoCo. These files should be a standard basic program saved with MSAVE.

CFG\_ENABLE\_DOS $20

Should DOS emulation be enabled or disabled.

CFG\_SHOW\_DATETIME $10

Should the RTC data and time be shown at startup.

CFG\_SHOW\_COMPILE $08

Should the firmware compile time be shown at startup.