STUDI/O Host interface specificationrev.0.9

This is the "synchronous transfer" approach. In this approach, a regular interrupt is posted by the board, and the host reads a fixed number of channels/samples and writes a fixed number of channels/samples.

The board has two operational states: Idle and Run. In Idle mode, all setup activity occurs, including reloading the FPGA, resynchronizing, changing I/O formats, etc. This can all be done from a relatively high level on the host, as none of it is time critical. In Run mode, the board posts regular interrupts, and the ISR should be the *only* point of communication with the board. Fortunately, there are only five commands in this mode: Read Samples, Write Samples, Read Status, Write Parameters, Reset Error. In Run mode, higher software levels on the host communicate only via software structures updated and read by the ISR. This keeps it simple!

The monitor mixer continues to operate and send audio in both Idle and Run modes, as do the peak meters and timecode reader (from the optional backplate).

Also, the DSP code is 'dumb' about sample rates and synchronization. The host driver takes care of all switching of clocks, and synchronization procedures. The DSP returns status information and sample rate measurements, and the host has to continually monitor this status, and take steps if something is awry. This also means that the host code must be careful to only program valid combinations of data formats, clocks, routing, and serial port setups. These are noted in an Appendix below.

Host port setup

The 56301 host interface is quite flexible and must be configured appropriately. First, the Data Transfer Format must be set up:

- For bootup, HTF =01. The DSP boots via the host port, and will expect it's boot code (24 bit words) in the least significant 3 bytes of each 32-bit PCI write. The bootloader on the host must format the load file appropriately.
- For sample data transfers, the host can use one of the 24 bit modes, depending on the demands of the application.

Bootup and initialization

The 56301 on the Studi/o board comes out of power-on reset in mode 4, which is Host Bootstrap PCI Mode (32 bit). This is due to the MOD pins being permanently wired for this mode (the IRQx's aren't being used for anything). The bootstrap ROM program reads 24-bit words encapsulated in 32-bit PCI transfers (target writes):

- First, it reads the number of words in the program, and the starting address in P space.
- Then it reads the program words (consecutive, obviously).
- Finally, it executes a Jump to the starting address of the program, and normal execution begins.

The program loaded is BEAVIS, which is the self-contained driver code for the board. This code contains all the necessary functions to further initialize the board, test the onboard hardware, interact with the host, and perform all required functions for the operation of the board (like audio stuff – neato!)

NOTE that on rev-B copper and newer, an EEPROM has been added. Once properly programmed at the factory, the DSP boots up in EEPROM mode, which allows it to program some volatile PCI registers on rev-B and newer 56301 chips, allowing us to properly identify the board to the system as a Sonorus product. However, the EEPROM boot stub then performs the exact same boot loading steps as the non-EEPROM version. Thus, except for the initial factory programming, booting proceeds the same. Also, note that the EEPROM code leaves a 'cookie' in the STATUS_ERROR word for the host to read: this code identifies the copper rev, which allows the bootloader to load the proper FPGA fusemaps (since there are minor changes on the copper which need appropriately modified FPGAs).

The host is required to interact with the board (via the 56301) in many cases, and downright babysit the damn thing in others. The mechanism is the Host Interface within the 56301. This provides us with a facility to dispatch up to 81 different commands and have six unidirectional general-purpose flags, three to the DSP core and three from it. We will use these to set up automatic data transfers for streaming audio, as well as attending to housekeeping functions and handling exception conditions. If you haven't read (or at least skimmed) the two Motorola red books (56300 FM and 56301 UM), you better read `em before moving on.

General Operation

The basic things you need to do with the board are: Play audio, Record audio, Check status, and Update parameters. It's expected that the board can maintain audio streaming of 16 record channels and 16 play channels simultaneously with status and updates. To this end, the process of transferring data is as streamlined as possible, with an eye toward eliminating every possible communication overhead.

For now, the model of the host driver is that it consists of a core part, which actually communicates with the board, and up to 16 isonchronous audio stream drivers. I say that since the OS threads are basically separate and asynchronous for each stream driver, but the board operates at one basic sample rate and therefore the data rate itself forces synchronization. The audio stream drivers connect to the core driver.

The core part sets up the board, and controls its behavior. There are three basic states of the board:

1. Not initialized. This is the power up, and reset condition. The board is basically dead, waiting for operative code and FPGA programming to be loaded to it. If you want to reset the chip, you can tell if it's in this mode by sending it the first word (program start address). If it's 'eaten', then it's in this mode, and you can keep 'feeding' it. Otherwise, you must issue the Reboot Host Command, and resend the first word.

2. Idle. In this state the monitor mixer, peak meters, and timecode reader are running, but no interrupts are being posted and no audio data is transferring to the host. The host executes command interrupts (Host Commands) to configure the running-state setup parameters. The host must be ***sure*** to initialize, synchronize, and start up the serial streams (and all the I/O for that matter) and program the FPGA before going to the Running state. Also in this mode, HF5 is toggled at the sample rate, to allow synchronous 'starting' (transitioning to Run mode) of multiple boards.

3. Running. The board is interrupting the host at regular intervals. There is a predefined sequence of interrupt - read status, read samples, write samples, write parameters.. The host controls the reads and writes via host commands. Each operation has it's own host command. The Write Samples command must be executed (or else the record inputs will loop through to the play outputs!), and the Read Status is recommended, but the other two are optional, and can be used as needed.

Thus the core driver sets up the board with the *following sequence of operations*:

1. Initializes the board. This is a one-time operation done when the OS boots up. Leaves the board in the Idle state, with the FPGA programmed for the particular I/O requirements selected by the user.

2. Synchronizes the board. Any time the user changes the data formats or clocks or routing, or the board loses sync (e.g. disconnected input), we go thru this step. First, the FPGA may be reprogrammed for a different I/O setup. Then the serial ports must be restarted. Then, from the idle state, the host checks the measured sample rate (via status group reads) and various other status, and writes to the FPGA to change it's operation. Once the sample rate has stabilized and the PLL's, etc. are running error-free, the host it can bring the board to the Run state. Also, any host-side driver setup should be done either here or step 1 (e.g. memory allocation, etc.)

3. Enters Running state. Here, the board issues a regular interrupt, and transfers a fixed amount of data. The driver has configured itself to move this as quickly as possible to defined areas of memory, and interface to the application-level audio buffers.

If the user desires to change the board configuration, i.e. I/O format, number of record or play channels, routing, etc. then the driver forces (via host command) the board back into the Idle state, and updates the setup information, and reenters the Run state.

Since the audio is read out as 64 contiguous samples for each channel, from 1-16, the core driver can use efficient block-moves to update independant channel buffers, necessitating only 16 pointer-changes (for play, 16 for record) to do so, and no extra data-copy operations.Also, the core driver is the synchronization focus of the system, as it must manage these isosynchronous channel buffers.

Play and Record sequences

Playing and recording happen hand-in-hand, since the onboard memory is used for both play and record buffers. The memory is split in half and used as a ping-pong buffer. While one half is being serviced by the DSP, the host updates the other half. Thus, when an interrupt comes, the host reads the buffer first, fetching the record samples. It then writes over the same area with the play samples. Meanwhile, the DSP plays the samples, and one by one replaces them with recorded samples as they come in.

ADDITION: A separate output channel 17/18 have been added, with it's own buffer and host command, to allow separate stereo monitoring. This stereo pair is summed with the normal monitor mix (just before the output level) and send to the monitor output.

Currently we're using 2048 samples total for this. Thus each half of the ping-pong is 1024 samples. This gives 64 samples to each of 16 channels, or 1.3mS at 48KHz. That will be our interrupt rate.

Two of the setup parameters are #play channels and #record channels. This dictates how much of the memory is actually read and written. However, we're stuck at 64samples/channel, allowing independent variation of record and play channels.

The board is configurable, with each of the two inputs and two outputs changeable between two channel operation (SPDIF-2) and eight channel operation (ADAT). Thus the number of channels can be four, ten, or sixteen. The core driver will 'record' and 'play' all channels configured. However, the data may be sent to the bit bucket if there is no channel buffer destination. Same with 'play' data -- unused channels should be sent zeros to mute them. This approach allows on-the-fly punch-in's and mute/unmutes without reconfiguring or even stopping the board.. Also, the core driver will handle channel mapping between driver channels and I/O channels. The board just feeds the channels through.

Here’s a table of the hardware channel mappings, given the four possible configurations. Note that this table applies equally to input and output. Also, the driver should have another layer of mappings, allowing any WaveAudio channel to be mapped to any hardware channel.

A *SPDIF* ***B*** *SPDIF*	***A*** *SPDIF* ***B*** *ADAT*	***A*** *ADAT* ***B*** *SPDIF*	***A*** *ADAT* ***B*** *ADAT*
1&2 ***A****, 3&4* ***B***	*1&2* ***A****, 3-10* ***B***	*1-8* ***A****, 9-10* ***B***	*1-8* ***A****, 9-16* ***B***

The **monitor bus** is a stereo mix of all 16 sixteen play streams (with a L and R level for each) and all 16 record streams (typically the record streams are only on in the stereo mix during overdubbing – this switch over is handled by the core driver, by sending new parameters). This monitor bus is generated onboard (requiring no extra bus bandwidth or CPU power) and goes to the D/A converter. Two other destinations can be specified for the monitor mix; these can be any four output channels. Usually this is used for DAT outputs, but could be used with the ADAT format as well. Additionally, a stereo sample stream from the Host is mixed into the monitor outputs, summed after the master level controls of the stereo mix.

So, when the board is in the Run mode, the driver action progresses as follows:

1. Receive interrupt from board
2. Read block of status information.
   - Issue Read Status host command.
   - Read the status.
3. Possibly set Host Port Transmit *and* Receive Data Format to 0, which is packed-16 mode if in 16-bit sample resolution mode(or a 24-bit mode if in that resolution). Also, HI should be in ‘Prefetch’ mode, to make reading more efficient.
4. Read block of record samples.
   - Issue Read Samples host command (this sets up the DSP to send the recorded audio over).
   - Read Nsamples. We have also extra host commands to read only 4 channels at a time, to cut down PCI bus usage when only a few record channels are used (since reads are less efficient than writes by a factor of 2-3).

5. Set Host Port Receive Data Format to 1, 2, or 3, depending on how you want the status info aligned within 32-bit words.

6. Write block of play samples (note Transmit Format (16-bit packing) already set).
7. Write block of montor samples for play (optional).

- Issue Write Samples host command.
- Write block of samples (N).

8. Set Host Port Transmit Data Format to 1, 2, or 3, depending on how the parameter info is aligned.
9. Write block of parameters.

- Issue Write Parameters host command.
- Write parameter block.

10. Issue Acknowledge Interrupt Host Command
11. Return from interrupt.

Status and Parameters

.

The **FPGA group** includes:

- Up to 32 bytes corresponding to the 32 available bytewide registers in the FPGA. These are written and read with special host commands, where a 32-bit command word is formed with a data byte and address concatenated for each register. These registers are stored directly within the FPGA and control it's functionality. They are not stored within DSP memory, except that certain status information from the FPGA is automatically inserted into the Status Group by the DSP, facilitating efficient transfer to the host.

The control registers within the FPGA are particular to each I/O format configuration. See the appendix below for details.

The board **setup group** includes:

- Format of each serial port. The host sets this to correspond to the I/O configuration set up via the FPGA group. The ADAT format has a frame of eight 24-bit words for both input and output, while the AES format has a frame of two 24-bit words.
- Also specified (in the serial format word) is data shift. The sample data can be shifted left or right by 8 bits in order to use PCI mode 0, where 32-bit PCI words become two 16-bit words right-justified within 24-bit words. Note that this is a global setting.

- Total play channels N(o), and total record channels N(i).
- Monitor destinations 1 & 2 for Left and Right.
- interrupt timer interval, or sample block size. Note that this value is preprogrammed by the DSP; it's here only for the host to read.
- Programmable sample rate counter value. This allows variable sample rates, with the sample rate being Fp/(2*(C+1)), where Fp is the processor clock speed and C is the counter value.

If the **setup group** is read as an array of 32-bit int's (sign extended), then its structure is as follows:

```
SETUP[0] = FORMAT;   //bit vector
SETUP[1] = NPLAY;    //number of play channels
SETUP[2] = NREC;     //number of record channels
SETUP[3] = MON1L;    //extra monitor destination #1 left
SETUP[4] = MON2L;    //extra monitor destination #2 left
SETUP[5] = MON1R;    //extra monitor destination #1 right
SETUP[6] = MONI2R;   //extra monitor destination #2 right
SETUP[7] = MSCALE;   //scale factor for peak meters reported in STATUS
SETUP[8] = TINT;     //sample block size per interrupt per channel (read only)
SETUP[9] = TCPR;     //programmable sample rate (timer) divider value
```

The monitor destination words have the following format:

Bit[3..0]	channel number of destination, on a particular output (A or B)
Bit[8]	enable. When set, enables this destination
Bit[9]	output A/B select. When set, steers this destination to numbered channel on output B. Cleared, destination is output A.

The FORMAT word has the following format:

Bit[1,0]	SIN0	in Aformat
Bit[3,2]	SOUT0	out A format
Bit[5,4]	SIN1	in B format

Bit[7,6]	SOUT1	out B format
Bit[8]		
Bit[9]	TCEN	enable timecode updates of sample counter (when set)

Formats defined so far are:

0	ADAT (8 channels)
1	SPDIF (2 channels)

The **status group** includes:

- Error conditions. If any errors occurred, DSP flags & codes them, e.g. overrun.
- I/O interface status (PLL lock, input good, etc. -- see appendix)
- Measured sample rate(s)
- DSP code execution time (peak). For internal use only! This tells us how much time we have to spare onboard. This can be compared directly to the measured sample rate, both are in DSP clock ticks (ticking once per two clocks).
- Sample ping-pong buffer address. For internal use only! This reflects the ‘side’ of the ping-pong buffer the DSP is currently processing.
- Sample counter. This counter is constantly increased by the DSP when in Run mode. Also, if the timecode reader is enabled, valid timcodes are forced into this location, then the DSP continues to count from there.
- Raw timcode value. The most recently decoded timecode, not incremented.
- Peak value of each channel (x 34) of that block of samples; inputs, outputs, and monitor. These are the direct 24-bit peak sample values as present on the inputs or outputs. They should be converted to decibels for standard meter display. These are cleared every interrupt, so they should be read every interrupt. This facilitates the host driver maintaining multiple independant and asynchronous peak meter displays. (Note that in Idle mode, these are not cleared, and must be cleared by writing the status group.)

If the **status group** is read as an array of 32-bit int’s (sign extended), then its structure is as follows:

STATUS[0] = ERR; //error bit vector

STATUS[1] = LTIME; //loop execution time
STATUS[2] = IOSTAT; //status of I/O interface
STATUS[3] = x; //for debugging
STATUS[4] = SR; //sample rate (measured)
STATUS[5] = PING; //p.p. buffer address (debugging)
STATUS[6] = SCNT_lo //low 24 bits of sample counter
STATUS[7] = SCNT_hi //hi 24 bits of sample counter (only bottom 8 used with timecode enabled)
STATUS[8] = TCNT_lo //low 24 bits of timecode
STATUS[9] = TCNT_hi //hi 24 bits of timecode (only bottom 8 used)
STATUS[10..26] = METERS_o; //16 channels output meters
STATUS[27..43] = METERS_i; //16 channels input meters
STATIS[44..45] = METERS_m; //2 channels monitor mix output meters

The ERR bit vector has the following format:

Bit[0]	No clock (clock <10KHz)
Bit[1]	Slow clock (clock <31KHz)
Bit[6]	REV-D copper (=1) – check on boot
Bit[7]	EEPROM present (=1) – check on boot
Bit[21]	Generic error (read STATUS.x for error code)
Bit[22]	DRAM test error (read STATUS.x for address)
Bit[23]	Real time violated (new interrupt posted before last one cleared)

The IOSTAT bit vector has the following format:

Bit[7..0]	Read from FPGA (see appendix A)
Bit[8]	timecode status (1=paused, 0=running)

The **parameters group** includes:

- L & R level for each of N(i) input and N(o) output samples to the stereo monitor mix output. These are ‘target levels’, in other words, the DSP linearly ‘fades’ between the last value and this value, reaching the new value at the last sample of the block. This de-clicking can be augmented by the driver calculating longer ‘gain trajectories’ and feeding the intermediate values to the DSP every block. (The first revision doesn’t linearly fade, it clicks!) These are 24-bit fractions with 6 bit integer part and 18 bit fractional part, by which the levels are multiplied before summing. (NOTE: The levels should be shifted 6 bits to give a unity gain.) Thus, they represent a linear scaling between (+64,-64). It is recommended that they never go over this 1/64 scale, as the mixer output is multiplied by 64 (this is to avoid clipping on the internal summing nodes of the mixer). The user should be presented with the equivalent dB attenuation, of course.

- Master Monitor levels (L & R), which scales the mixed stereo monitor. Since the Monitor sums 24-bit samples, it’s maximum is 6 bits (64x) over the input levels (including the extra pairs), so this digital scaling allows us to avoid clipping.
- D/A (analog monitor) attenuations. A byte for each L & R, plus a control BYTE (DACON from the FPGA) are packed here. The L & R range from 0 (0 dB) to 63 (-63dB) to mute (64). Values 64-127 cause the output to mute. Note that the attenuator is analog, and causes the noise floor of the D/A to also be attenuated.

Channel Phase. Two 24-bit words store (in the 16 LSBs each) a phase invert bit for each input and output. This phase inversion happens directly at the I/O port. 0=0 degrees, 1=180 degrees. Mainly this is to accommodate the original ADATs, which recorded signals out of phase on tape.

If the **parameters group** is read as an array of 32-bit int’s (sign extended), then its structure is as follows:

PARM[31..0] = output monitor levels, as left-right pairs, for each of 16 outputs

PARM[33..32] = master monitor levels, left first then right

PARM[65..34] = input monitor levels, as left-right pairs, for each of 16 inputs

PARM[66] = D/A control word

PARM[67] = output phases (LS 16 bits) 1=invert 0=no invert.

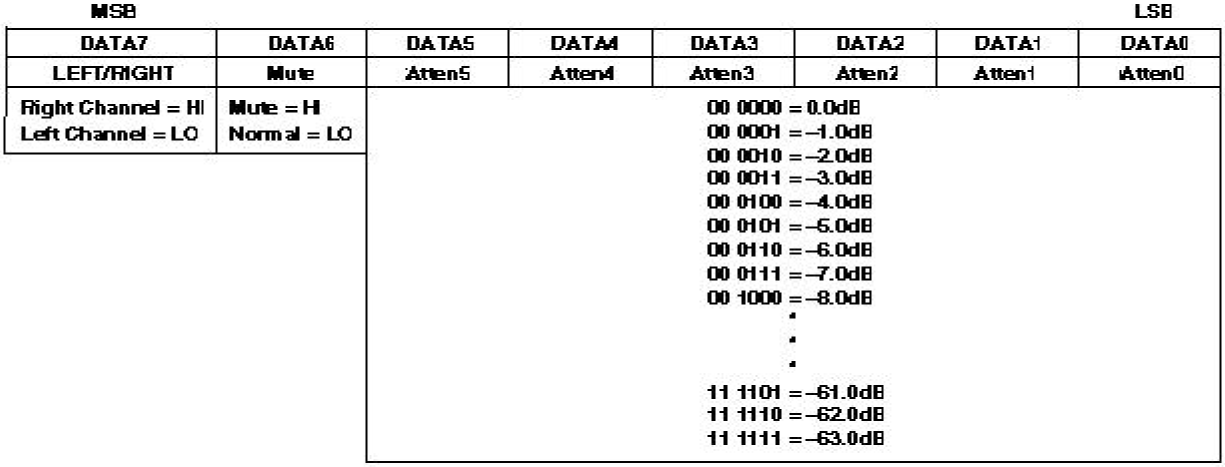
PARM[68] = input phases (LS 16 bits) 1=invert 0=no invert.

The D/A control word has the following format:

Bits[2..0]	set to zero

Bit[3]	mute = 1
Bit[4]	resolution (1=18 bits, 0=16 bits)
Bit[5]	
Bit[6]	don’t care
Bit[7]	Fs range (1=34KHz and above, 0=34KHz and below)
Bits[15..8]	D/A attenuate byte left (set bit15=0)
Bits[23..16]	D/A attenuate byte right (set bit23=1)

The D/A attenuate bytes are formatted as follows:



MSB LSB

DATA7	DATA6	DATA5	DATA4	DATA3	DATA2	DATA1	DATA0
LEFT/RIGHT	Mute	Atten5	Atten4	Atten3	Atten2	Atten1	Atten0
Right Channel = HI Left Channel = LO	Mute = HI Normal = LO	00 0000 = 0.0dB 00 0001 = –1.0dB 00 0010 = –2.0dB 00 0011 = –3.0dB 00 0100 = –4.0dB 00 0101 = –5.0dB 00 0110 = –6.0dB 00 0111 = –7.0dB 00 1000 = –8.0dB . . . 11 1101 = –61.0dB 11 1110 = –62.0dB 11 1111 = –63.0dB					

<u>HOST FLAGS</u>

One of the host interface flags, HF3, indicates the state of the board (set by the DSP in response to host commands):

HF3 = 0	**HF3 = 1**
idle state (or not initialized state)	**running** state

This will come in handy, as you’ll see!

Another host interface flags, HF4, indicates that the DSP is currently in the middle of a host command:

HF4 = 0	**HF4 = 1**
Ready for a command	Currently processing host command (sending/waiting for data)

Yet another host interface flag, HF5, is used by the host for syncronous multiboard starts. The DSP mirrors the sample clock to this host flag, so the host can poll it and start all cards on a falling edge, having an entire sample period to get things going:

HF5
Sample clock (30-50KHz square wave)

Note that if no sample clock is running (e.g. FPGA is not programmed) then this pin will not toggle. THIS HF IS ONLY IN OPERATION DURING **IDLE** MODE.

Host Commands

Here are the host commands, listed by the actual number used to dispatch them, with descriptions. Of the two types, fast indicates that the host need not wait for any acknowledgment, ack means it does. Check the following section (Host Command Details) for the structure of the extra data. Note that fast doesn't correspond to the 56301's fast interrupt. Also, note that there are actually 128 Host Commands available to be dispatched, but use of the pre-assigned commands isn't generally recommended, with the exception of RESET ($00) and Debug Request ($03).

Command # (hex)	Description / Function	Type
06	Enter idle state (muting audio)	fast
07	Enter running state (enabling interrupt)	fast
1E	Send setup group	ack
1F	Readback setup group	ack
26	Send status group	ack
27	Readback status group	ack
2D	Send byte to FPGA (with address)	fast
2E	Get byte from FPGA (from address)	ack
2F	Reprogram FPGA (pull reprogram pin of Altera part & send program)	ack
39	Send parameter group	ack
3A	Readback parameter group	ack
3B	Stop serial streams	fast

3C	Start serial streams	fast
3D	Clear (acknowledge) host interrupt	fast
3E	REBOOT	fast
3F		
40		
41		
42		
43	Read Samples	ack
44	Write Samples	ack
45		
46		
47		
48		
49		
4A	Reset Error	fast
4B	Write byte to EEPROM (with address)	ack
4C	Read byte to EEPROM (with address)	ack
4D	Write Samples to Monitor	ack
4E	Set HINT	fast
4F	Read Samples0 - channels 0-3 (block 0)	ack
50	Read Samples1 - channels 4-7 (block 1)	ack
51	Read Samples2 - channels 8-11 (block 2)	ack
52	Read Samples3 - channels 12-15 (block 3)	ack
53		
54		
55		
56		
57		
58		
59		
5A		
5B		
5C		
5D		
5E		
5F		
60		
61		
62		
63		

64		
65		
66		
67		
68		
69		
6A		
6B		
6C		
6D		
6E		
6F		
70		
71		
72		
73		
74		
75		
76		
77		
78		
79		
7A		
7B		
7C		
7D		
7E		
7F		

Host Commands Details

When the driver issues a command, it first checks HF4 to make sure a previous DMA-based host command isn't still pending (a serious error!), waits until the HI is clear (TRDY bit), and the writes it's parameter (if any) to the Host Rx register, and then issues the host command. This way the host command has the parameter immediately available. If it needs an ack, that may be just reading a word or two back, or it may be another series of handshakes, or block transfers of data. Each command is different.

Note that the board should be the idle state for the FPGA commands, especially Reprogram.

- **Enter idle state**

Parameter: none

Ack: none

Function: Mutes all the outputs, and disables interrupt, then waits for host commands.

- **Enter running state**

Parameter: none

Ack: none

Function: Enables interrupt, and enters real-time loop, servicing samples & processing stuff, etc.

- **Send setup group**

Parameter: none

Ack: None. Just start sending the setup group (block). Might get a couple wait states while the DSP sets up the transfer.

Function: sends setup parameter block to the board, which overwrites the old one. It's a predefined size and sits in a predefined location. . Sets HF4 to indicate a transfer is in progress. The DMAC clears HF4 when done.

- **Readback setup group**

Parameter: none

Ack: None. Just starts returning the setup group. Again, waits will be incurred on the first read while the DSP sets up the transfer. Again, predefined size. . Sets HF4 to indicate a transfer is in progress. The DMAC clears HF4 when done.

- **Send status group**

Parameter: none

Ack: None. Just start sending the status group. Might get a couple wait states while the DSP sets up the transfer.

Function: Sends status group to the board, which overwrites the old one. It’s a predefined size and sits in a predefined location. . Sets HF4 to indicate a transfer is in progress. The DMAC clears HF4 when done.

- **Readback status group**

Parameter: none

Ack: None. Just start reading the parameter group. Might get a couple wait states while the DSP sets up the transfer.

Function: Gets the parameter group from the board. It’s a predefined size and sits in a predefined location. . Sets HF4 to indicate a transfer is in progress. The DMAC clears HF4 when done.

- **Send parameter group**

Parameter: none

Ack: None. Just start sending the parameter group. Might get a couple wait states while the DSP sets up the transfer.

Function: sends parameter group to the board, which overwrites the old one. It’s a predefined size and sits in a predefined location. . Sets HF4 to indicate a transfer is in progress. The DMAC clears HF4 when done.

- **Readback parameter group**

Parameter: none

Ack: None. Just start reading the parameter group. Might get a couple wait states while the DSP sets up the transfer.

Function: returns the parameter group from the board. It’s a predefined size and sits in a predefined location. . Sets HF4 to indicate a transfer is in progress. The DMAC clears HF4 when done.

- **Send byte to FPGA (with address)**

Parameter: bottom byte is byte to send to FPGA, next byte is address (actually only 5 lsb's are used) since the FPGA appears at 32 addresses in the DSP memory space.

Ack: none

Function: Sends byte right straight to the FPGA, no questions asked!

- **Get byte from FPGA**

Parameter: second to bottom byte is address (actually only 5 lsb's are used) since the FPGA appears at 32 addresses in the DSP memory space.

Ack: returned byte from FPGA in lowest byte.

Function: Gets byte right straight from the FPGA, no questions asked!

- **Reprogram FPGA (pull reprogram pin of Altera part)**

Parameter: none

Ack: none. Send fuse programming bytes, one at a time in the low byte of each word, with the second-lowest byte zero except for the last byte of the program, which should be $FF.

Function: toggles program pin on FPGA, causing it to go into program mode, then receives the programming info from the host. *Will not exit until done programming*, since timing is critical for this operation. . Sets HF4 to indicate a transfer is in progress. Clears HF4 when done.

- **Stop serial ports**

Parameter: none

Ack: none

Function: halts serial port DMA.

- **Start serial ports**

Parameter: none

Ack: none

Function: programs serial ports according to data in setup group, and starts serial DMA. Note that the FPGA must be set up (clock source, etc) prior to issuing this command.

- **Clear (ack) interrupt**

Parameter: none

Ack: none

Function: causes DSP to deassert interrupt A line. Probably the first thing to do in the interrupt service routine on the host!

- **REBOOT**

Parameter: none

Ack: none

Function: causes DSP reboot. That is, it jumps to it’s internal boot-loader EPROM, which then expects a new program to be loaded via the Host Interface, as per Motorola spec.

- **Read Samples**

Parameter: none

Ack: none. Just start reading the sample block. Might get a couple wait states while the DSP sets up the transfer. Function: sets up HI DMA to start sending current sample block (pingpong) buffer. Sets HF4 to indicate a transfer is in progress. The DMAC clears HF4 when done.

- **Write Samples**

Parameter: none

Ack: none. Just start sending the samples. Might get a couple wait states while the DSP sets up the transfer. Host must keep track of block count. Function: sets up HI DMA to start receiving current sample block (pingpong) buffer. . Sets HF4 to indicate a transfer is in progress. The DMAC clears HF4 when done.

- **Reset Error**

Parameter: NONE

Ack: none.

Function: Clears Setup.Error.

- **Write byte to EEPROM**

Parameter: bottom byte is byte to send to EEPROM, next 2 bytes are address (actually only 15 lsb's are used) since the EEPROM appears at 32K addresses in the DSP memory space.

Ack: none.

Function: performs special cycle to write the byte to the EEPROM. *Will not exit until done programming.* **NOTE: DSP expects the EEPROM to be organized in specific sections: see Appendix for details. Especially note that it reads BOOT code from the beginning of the EEPROM, if this is overwritten, general mayhem will result!**

- **Read byte to EEPROM**

Parameter: bottom byte is a don't care -- next 2 bytes are address (actually only 15 lsb's are used) since the EEPROM appears at 32K addresses in the DSP memory space.

Ack returned byte from EEPROM in lowest byte

Function: reads back a byte of data from the EEPROM.

- **Write Samples to Monitor**

Parameter: none

Ack: none. Just start sending the samples. Might get a couple wait states while the DSP sets up the transfer. Host must keep track of count.

Function: Send block of 64 samples each of left then right channels which is directly summed into the monitor bus. Ping-pong behavior the same as the regular sample buffers.

- **Set HINT**

Parameter: none

Ack: none.

Function: Sets the HINT line. Used to sense interrupt during manual interrupt routing.

- **Read Samples0 (channels 0-3)**

Parameter: none

Ack: none. Just start reading the sample block. Might get a couple wait states while the DSP sets up the transfer. Function: sets up HI DMA to start sending current sample block (pingpong) buffer. Sets HF4 to indicate a transfer is in progress. The DMAC clears HF4 when done.

Function: read 4-channel block of samples. Recording is 2-3 times slower than playing, so we can gain some bus efficiency when we're only recording a few channels.

- **Read Samples1 (channels 4-7)**

Parameter: none

Ack: none. Just start reading the sample block. Might get a couple wait states while the DSP sets up the transfer. Function: sets up HI DMA to start sending current sample block (pingpong) buffer. Sets HF4 to indicate a transfer is in progress. The DMAC clears HF4 when done.

Function: read 4-channel block of samples. Recording is 2-3 times slower than playing, so we can gain some bus efficiency when we're only recording a few channels.

- **Read Samples2 (channels 8-11)**

Parameter: none

Ack: none. Just start reading the sample block. Might get a couple wait states while the DSP sets up the transfer. Function: sets up HI DMA to start sending current sample block (pingpong) buffer. Sets HF4 to indicate a transfer is in progress. The DMAC clears HF4 when done.

Function: read 4-channel block of samples. Recording is 2-3 times slower than playing, so we can gain some bus efficiency when we're only recording a few channels.

- **Read Samples3 (channels 12-15)**

Parameter: none

Ack: none. Just start reading the sample block. Might get a couple wait states while the DSP sets up the transfer. Function: sets up HI DMA to start sending current sample block (pingpong) buffer. Sets HF4 to indicate a transfer is in progress. The DMAC clears HF4 when done.

Function: read 4-channel block of samples. Recording is 2-3 times slower than playing, so we can gain some bus efficiency when we're only recording a few channels.

Appendix C: *FPGA formats*

2020.TTF: 2-ADAT in, 2-ADAT out.

Only register 0 should be directly written to: register 1 is updated via the DACON field of the Parameter Group.

				Bit				
Addr	7	6	5	4	3	2	1	0
0			clock 00 01 10 11	source 44.1K 48K PLL N/A	PLL 00 01 10 11	source optical A optical B timer 1 external		
1	D/A range (1=44.1 and above, 0=below)		optical range (1=long, 0=normal)	D/A resolution (1=18 bits, 0=16 bits)	D/A mute (=1)	CLAT	CCLK	CDATA

Writeable Registers

This readable register is passed back in the Status Group, so it shouldn't directly be read

				Bit				
Addr	7	6	5	4	3	2	1	0
0					T.C.	input B data error (=1)	input A data error (=1)	No ADAT sync.

Readable Registers

Format setup:

For 2020.TTF, the format word in the Setup Group should be:

0x100 for 24-bit audio mode.

Also in the Setup Group; NplayChannels=NrecChannels=0x10, as there are 16 channels of both input and output.

1111.TTF: 1-ADAT 1-DAT(SRC) in, 1-ADAT 1-DAT out.

Registers other than 1 should be written to set up FPGA: register 1 is updated via the DACON field of the Parameter Group.

				Bit				
Addr	7	6	5	4	3	2	1	0
0			clock 00 01 10 11	source 44.1K 48K PLL AES	PLL 00 01 10 11	source optical A N/A timer 1 external		
1	D/A range (1=44.1 and above, 0=below)		optical range (1=long, 0=normal)	D/A resolution (1=18 bits, 0=16 bits)	D/A mute (=1)	CLAT	CCLK	CDATA
2	byte 0 of AES/IEC channel status block							
3	byte 1 of AES/IEC channel status block							
4	byte 2 of AES/IEC channel status block							
5	byte 3 of AES/IEC channel status block							

Writeable Registers

The recommended settings for the AES status are:
byte0 = 04h, sets mode to consumer and copy bit to "No Copyright Present"
byte1 = 03h, sets category code and "generation bit" to original magnetic recording
byte2 = 00h

byte3 = 00h for 44.1kHz
byte3 = 02h for 48kHz
byte3 = 03h for 32kHz
byte3 has be set to match sample rate of audio being transmitted.

This readable register is passed back in the Status Group, so it shouldn't directly be read.

				Bit				
Addr	7	6	5	4	3	2	1	0
0		AES	chip	errors	T.C.		input A data error (=1)	No ADAT sync.
		0 0 0	No Error					
		0 0 1	Validity Bit	High				
		0 1 0	Confidence	Flag				
		0 1 1	Slipped	Sample				
		1 0 0	CRC Error	(PRO only)				
		1 0 1	Parity Error					
		1 1 0	Bi-Phase	Code Error				
		1 1 1	No Lock					

Readable Registers

Format setup:

For 1111.TTF, the format word in the Setup Group should be:

0x150 for 24-bit audio mode.

Also in the Setup Group; NplayChannels=NrecChannels=0xA, as there are 10 channels of both input and output.

0202.TTF: 0-ADAT 2-DAT in, 1-DAT(SRC) 1-DAT(no SRC) out.

This program has two SPDIF inputs, which must be locked together. Designed for stuff like the Lexicon 2020A/D, which outputs four channels, or some other 4-channel source. Output A goes thru the Sample Rate Converter, and has it's own selectable clock source. Output B follows the main clock source. Each output has it's own subcode byte for sample rate.

Registers other than 1 should be written to set up FPGA: register 1 is updated via the DACON field of the Parameter Group.

				Bit				
Addr	7	6	5	4	3	2	1	0
0	SRC clk 00 01 10 11	source 44.1K 48K PLL AES	clock 00 01 10 11	source 44.1K 48K PLL AES	PLL 00 01 10 11	source N/A N/A timer 1 external		N/A
1	D/A range (1=44.1 and above, 0=below)		optical range (1=long, 0=normal)	D/A resolution (1=18 bits, 0=16 bits)	D/A mute (=1)	CLAT	CCLK	CDATA
2	byte 0 of AES/IEC channel status block							
3	byte 1 of AES/IEC channel status block							
4	byte 2 of AES/IEC channel status block							
5	byte 3 of AES/IEC channel status block for output B							
6	byte 3 of AES/IEC channel status block for output A (SRC'd out)							

Writeable Registers

The recommended settings for the AES status are:
byte0 = 04h, sets mode to consumer and copy bit to "No Copyright Present"
byte1 = 03h, sets category code and "generation bit" to original magnetic recording
byte2 = 00h
byte3 = 00h for 44.1kHz
byte3 = 02h for 48kHz
byte3 = 03h for 32kHz
byte3 has be set to match sample rate of audio being transmitted (for each output).

This readable register is passed back in the Status Group, so it shouldn't directly be read.

				Bit				
Addr	7	6	5	4	3	2	1	0

0		AES 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0 1 0 1 1 1 0 1 1 1	chip No Error Validity Bit Confidence Slipped CRC Error Parity Error Bi-Phase No Lock	errors High Flag Sample (PRO only) Code Error				

Readable Registers

Format setup:

For 0202.TTF, the format word in the Setup Group should be:

0x155 for 24-bit audio mode.

Also in the Setup Group; NplayChannels=NrecChannels=0x4, as there are 4 channels of both input and output.

1120.TTF (aka 2020k.ttf): 1-ADAT 1-DAT(SRC) in, 2-ADAT out.

This format is specifically designed for the Korg 168RC digital mixing console.

Registers other than 1 should be written to set up FPGA: register 1 is updated via the DACON field of the Parameter Group.

				Bit				
Addr	7	6	5	4	3	2	1	0
0			clock 00 01 10 11	source 44.1K 48K PLL AES	PLL 00 01 10 11	source optical A N/A timer 1 external		
1	D/A range (1=44.1 and above, 0=below)		optical range (1=long, 0=normal)	D/A resolution (1=18 bits, 0=16 bits)	D/A mute (=1)	CLAT	CCLK	CDATA

This readable register is passed back in the Status Group, so it shouldn’t directly be read.

				Bit				
Addr	7	6	5	4	3	2	1	0
0		AES 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0 1 0 1 1 1 0 1 1 1	chip No Error Validity Bit Confidence Slipped CRC Error Parity Error Bi-Phase No Lock	errors High Flag Sample (PRO only) Code Error 	T.C.		input A data error (=1)	No ADAT sync.

Readable Registers

Format setup:

For 1120.TTF, the format word in the Setup Group should be:

0x150 for 24-bit audio mode.

Also in the Setup Group; NplayChannels=0x10 and NrecChannels=0xA, as there are 10 channels of input and 16 channels of output.

Appendix D: *Hostcom.h*