Version History

Version 1.01

Date: 10.01.2008

Changes: just text clarification; no content

Version 1.00

Date: 09.01.2008

Initial Version

Differences to MARIAN Trace PRO are yellow-highlighted

Introduction

This documentation covers the software interface to the Trace Alpha driver, which is useful for third party application software to control the Trace Alpha audio signals. It includes:

- Analog Input Level Control
- Mixer Channel Control
- Sum Channel Control
- Output Volume Control
- Output Signal Routing
- Signal Level Analysis

The software interface is delivered by the 32-Bit Windows™ DLL file *"talpifc.dll"*. Thus, it makes it very easy to use it in any 32-Bit-Windows™ based application software. The documentation includes a C++ language header file (*talpifc.h*) which contains all function prototypes and other declarations like constants and structures. Additional, this documentation comes with the Trace Alpha user manual which helps to understand how the Trace Alpha is working and how the Trace Alpha Manager software visualizes the cards functionality.

In the following, the Trace Alpha Control API is called *TALPIFC.*

Common Initialization/Finalization

The application software should first initialize the *TALPIFC* by use of the function `talp_Init`. Multiple clients of the interface are supported via the same DLL module.

If the software application finishes the use of *TALPIFC* it must call `talp_Done`.

It is essential for the client software to get the number of installed and active Trace Alpha PCI cards via `talp_GetNumCards` because the number of the card is a main identifier for many functions.

The Functions

talp_Init

```
BOOL TALPCALL talp_Init();
// initializes usage of TALPIFC
```

talp_Done

```
VOID TALPCALL talp_Done();
// finalizes usage of TALPIFC
```

talp_GetNumCards

```
LONG TALPCALL talp_GetNumCards();
// returns the number of installed, active Trace Alpha PCI cards
```

Handling Parameter Values

Every mixer parameter and also every output parameter can be read and set by use of two functions: `talp_mx_GetValue` and `talp_mx_SetValue`. Each parameter is uniquely identified by a set of ID's – the Card ID, the Channel Type ID, the Channel ID and the Parameter ID. Some parameters require also a sub parameter ID. Use the functions `talp_mx_GetValueEx` and `talp_mx_SetValueEx` to access these parameters.

The Card ID is a simple number beginning from 0 to the number of installed cards – 1. The number of installed cards can be easily determined through the function `talp_GetNumCards`. All other ID's are explained in the following sections.

The Functions

talp_mx_GetValue

```
LONG TALPCALL talp_mx_GetValue (
     LONG  crdid,
     LONG  typid,
     LONG  chnid,
     LONG  parid,
     PLONG pValue
);
// Gets the current Value of a specific mixer parameter
// Input:
//   crdid  = Card Id
//   typid  = Channel Type Id (see talp_typid defines)
//   chnid  = Channel Id (see talp_chnid defines)
//   parid  = Parameter Id (see talp_parid defines)
//   pValue = Pointer to a LONG variable for the result
// Output:
//   *pValue filled will the current parameter value
//   return < 0 ? Error : Success
```

talp_mx_GetValueEx

```
LONG TALPCALL talp_mx_GetValueEx (
     LONG  crdid,
     LONG  typid,
     LONG  chnid,
     LONG  parid,
     LONG  sparid,
     PLONG pValue
);
// Gets the current Value of a specific mixer parameter
// Input:
//   crdid  = Card Id
//   typid  = Channel Type Id (see talp_typid defines)
//   chnid  = Channel Id (see talp_chnid defines)
//   parid  = Parameter Id (see talp_parid defines)
//   sparid = Sub-Parameter Id (see talp_sparid defines)
//   pValue = Pointer to a LONG variable for the result
// Output:
//   *pValue filled will the current parameter value
//   return < 0 ? Error : Success
```

talp_mx_SetValue

```
LONG TALPCALL talp_mx_SetValue (
     LONG  crdid,
     LONG  typid,
     LONG  chnid,
     LONG  parid,
     LONG  Value
);
// Sets the Value of a specific mixer parameter
// Input:
//   crdid  = Card Id
//   typid  = Channel Type Id (see talp_typid defines)
//   chnid  = Channel Id (see talp_chnid defines)
//   parid  = Parameter Id (see talp_parid defines)
//   Value  = New Value for the Parameter
// Output:
//   return < 0 ? Error : Success
```

talp_mx_SetValueEx

```
LONG TALPCALL talp_mx_SetValueEx (
     LONG  crdid,
     LONG  typid,
     LONG  chnid,
     LONG  parid,
     LONG  sparid,
     LONG  Value
);
// Sets the Value of a specific mixer parameter
// Input:
//   crdid  = Card Id
//   typid  = Channel Type Id (see talp_typid defines)
//   chnid  = Channel Id (see talp_chnid defines)
//   parid  = Parameter Id (see talp_parid defines)
//   sparid = Sub-Parameter Id (see talp_sparid defines)
//   Value  = New Value for the Parameter
// Output:
//   return < 0 ? Error : Success
```

Mixer Control

Mixer Architecture

The Mixer of the Trace Alpha consists of 20 mono channels.

The first four channels control the audio signals of the input channels of the card. The next eight channels of the mixer channels control the audio signals of the playback channels of the card. “Playback channels” does not mean output channels. It really means the playback signals which any audio application sends to the drivers software. And the last eight channels control the signals laying on TDM BUS and which are used here as mixer input channels.

Which audio signal can be heard at a output depends on the Output Routing (see Output Control).

The signals of all mixer channels are mixed to eight individually sums – AUX1 … AUX6 and MASTER.

Mixer Channel Control

Each mixer channel is identified by an ID for its type as mixer channel and an individual channel ID.

Type ID (`talp_typid`)	Channel ID (`talp_chnid`)	Trace Alpha Device
0	0	Analog Input 1
0	1	Analog Input 2
0	2	Digital Input L
0	3	Digital Input R
0	4	Play 1
0	5	Play 2
0	6	Play 3
0	7	Play 4
0	8	Play 5
0	9	Play 6
0	10	Play 7
0	11	Play 8
0	12	TDM 1
0	13	TDM 2
0	14	TDM 3
0	15	TDM 4
0	16	TDM 5
0	17	TDM 6
0	18	TDM 7
0	19	TDM 8

See the appropriate type ID’s (`talp_typid_xxx`) and channels ID’s (`talp_chnid_xxx`) also in the *talpifc.h*.

Each mixer channel has a number of parameters. The following table shows these parameter.

<table>
<tr><th>Parameter ID
(talp_parid)</th><th>Name</th><th colspan="3">Value Range</th><th>Remarks</th></tr>
<tr><td rowspan="3">0</td><td rowspan="3">Source Select</td><td>Value</td><td colspan="2">Source</td><td rowspan="3">Digital Input Channel Only: Determines the physical audio connector for the channel.</td></tr>
<tr><td>0</td><td colspan="2">RCA Input</td></tr>
<tr><td>1</td><td colspan="2">CD Input</td></tr>
<tr><td>8</td><td>Reserved</td><td colspan="3"></td><td>Trace PRO: Gain Range; not available for Trace Alpha</td></tr>
<tr><td>9</td><td>Analog Gain</td><td colspan="3">0..0x0A400 == -INF ..+18 dB</td><td>For Analog Input Channels Only. Influences the recording level too</td></tr>
<tr><td>1</td><td>Digital Gain</td><td colspan="3">0..0x10000 == -INF ..+6 dB</td><td>Signal Level Pre-Adjustment. (Mixer only)</td></tr>
<tr><td>2</td><td>AUX Volume</td><td colspan="3">0..0x10000 == -INF ..+6 dB</td><td>Signal level for mix at AUX N. This parameter needs a sub parameter which specifies the AUX number</td></tr>
<tr><td>3</td><td>AUX Pre Fader</td><td colspan="3">1 || 0 == on || off</td><td>Determines whether or not the channel signal is mixed to the AUX N Bus pre (on) or post (off) the fader section. This parameter needs a sub parameter which specifies the AUX number</td></tr>
<tr><td>4</td><td>Mute</td><td colspan="3">1 || 0 == on || off</td><td>Mutes (On) the channel signal at the AUX and Master Busses.</td></tr>
<tr><td>5</td><td>Solo</td><td colspan="3">1 || 0 == on || off</td><td>Set the channel signal at the Master Sum to Solo (On).</td></tr>
<tr><td rowspan="5">6</td><td rowspan="5">Master PAN</td><td>Value</td><td>Left</td><td>Right</td><td rowspan="5">Determines the signal level of the channel at the left and right channel of the Master sum.</td></tr>
<tr><td>0</td><td>+6 dB</td><td>-INF</td></tr>
<tr><td>0x08000</td><td>0 dB</td><td>0 dB</td></tr>
<tr><td>0x10000</td><td>-INF</td><td>+6 dB</td></tr>
<tr><td></td><td colspan="3"></td></tr>
<tr><td>7</td><td>Master Volume</td><td colspan="3">0..0x10000 == -INF ..+6 dB</td><td>Signal level for the mix at the Master Sum</td></tr>
</table>

Generally, all the values for dB ranges are like PCM sample values. The maximum value is +6 dB. Every *"shift right"* operation of the parameter value decreases the dB value by 6 dB. Thus, the following conversion routines apply:

Parameter Value to dB Value:

```
dBValue = Log2 (ParamValue/0x10000)*6 + 6;
```

dB Value to Parameter Value:

```
ParamValue = Round (Power (2, (dBValue-(-90))/6));
```

The Analog Gain Inputs is set directly in the AD converter. Thus, another value to dB conversion applies. Only the bits D8..D15 are significant. Please see the conversion table in the file *ek4620a.pdf* page 33. Additional, when the gain of the analog inputs is manipulated, this also influences the analog recording level, not only the mixer input level.

Sum Channel Control

Each Sum channel is identified by an ID for its type as sum channel and an individual sum channel ID.

Type ID (talp_typid)	Channel ID (talp_chnid)	Sum Channel
1	0	Master Left
1	1	Master Right
1	2	Aux 1
1	3	Aux 2
1	4	Aux 3
1	5	Aux 4
1	6	Aux 5
1	7	Aux 6

See the appropriate type ID's (`talp_typid_xxx`) and channels ID's (`talp_chnid_xxx`) also in the *talpifc.h*.

Each sum channel has a parameter for the level adjustment of the audio mix. The following table shows this parameter and it's value range.

Parameter ID (talp_parid)	Name	Value Range	Remarks
0	Volume	`0..0x8000 == -INF .. 0 dB`	Sets the audio mix level for sum channel.

The conversion from dB value to parameter value and back should proceed like described in *"Mixer Channel Control".*

Output Control

Output Architecture

The main task of the output control is the output signal routing and the control of the output signal level. Each channel of the output control is assigned to a physical output of the Trace Alpha.

Output Channel Control

Each output stereo channel is identified by an ID for its type as output channel and an individual channel ID.

Type ID (talp_typid)	Channel ID (talp_chnid)	Trace Alpha Device
2	0	Analog Output 1-2
2	1	Digital Output
2	2	TDM Bus Output 1-2
2	3	TDM Bus Output 3-4
2	4	TDM Bus Output 5-6
2	5	TDM Bus Output 7-8

See the appropriate type ID's (`talp_typid_xxx`) and channels ID's (`talp_chnid_xxx`) also in the *talpifc.h*.

Each output channel has a number of parameters, shown in this table.

<table>
<tr><th>Parameter ID (talp_parid)</th><th>Name</th><th>Value Range</th><th>Remarks</th></tr>
<tr><td>0</td><td>Source Select</td><td><table><tr><td>Value</td><td>Source</td></tr><tr><td>0</td><td>Play 1-2</td></tr><tr><td>1</td><td>Play 3-4</td></tr><tr><td>2</td><td>Play 5-6</td></tr><tr><td>3</td><td>Play 7-8</td></tr><tr><td>4</td><td>Play 9-10</td></tr><tr><td>5</td><td>Play 11-12</td></tr><tr><td>6</td><td>Play 13-14</td></tr><tr><td>7</td><td>Play 15-16</td></tr><tr><td>8</td><td>Analog Input</td></tr><tr><td>9</td><td>Digital Input</td></tr><tr><td>12</td><td>TDM 1-2</td></tr><tr><td>13</td><td>TDM 3-4</td></tr><tr><td>14</td><td>TDM 5-6</td></tr><tr><td>15</td><td>TDM 7-8</td></tr><tr><td>16</td><td>Master Sum</td></tr><tr><td>17</td><td>Aux 1-2</td></tr><tr><td>18</td><td>Aux 3-4</td></tr><tr><td>19</td><td>Aux 5-6</td></tr></table></td><td>Determines the source signal which should be used for the output. Any of the available signals can be used – input, playback or sum signals</td></tr>
<tr><td>1</td><td>Mute</td><td>1 || 0 == On || Off</td><td>Mutes the Output (on)</td></tr>
<tr><td>2</td><td>Volume Left</td><td>0..0x8000 == -INF .. 0 dB</td><td>Controls output level left channel</td></tr>
<tr><td>3</td><td>Volume Right</td><td>0..0x8000 == -INF .. 0 dB</td><td>Controls output level right channel</td></tr>
<tr><td>4</td><td>TDM On Bus</td><td>1 || 0 == On || Off</td><td>For TDM Output Channels only: Switches the signal on the Bus or not</td></tr>
<tr><td>5</td><td>Reserved</td><td></td><td>Trace PRO: Output Level Range; Not available for Trace Alpha</td></tr>
<tr><td>6</td><td>Format</td><td>1 || 0 == Non-Audio || Audio</td><td>For Digital Output Channels Only: Sets the according S/PDIF channel state</td></tr>
</table>

The conversion from dB value to parameter value and back should proceed like described in *“Mixer Channel Control”*.

Audio Level Analysis

The Trace Alpha hardware analyzes the level of every audio signal available. Thus, any client software is able to get the results of the level analysis for its own purposes – mainly for the implementation of level meters.

How it works

The Trace Alpha hardware stores level values in a special part of the card memory. But before a level value is stored, the hardware compares it with the previous stored value which is only overwritten, if the new value is higher than the older. The stored value is reset to zero, if the driver software reads the value from the hardware. This way, the driver software always gets the maximum level which occurred between two read operations.

The driver software does the same for registered client software. It reads the level value from the hardware compares it with the last stored, client related level value and stores it only, if the client level value is less than the new level value. The client related level value is reset to zero, if the client software read its level value.

In this manner the hardware works with every sample, the driver does it approximately every 1 ms and it is up to the client software to get the level value in intervals which are appropriate for the clients purpose.

The implementation of a level meter can be done very easily. The client software could have 2 processes. The first process gets the level and displays it only if the actual displayed level display is less than the new value. The second process simply decreases the level display by a determined dB value in a determined time.

The Functions

Structure TALP_CHNLEVELINFO

```
typedef struct {
  LONG   Pre;          // Source Level Pre Gain
  LONG   PostGain;     // Level measured after Gain
  LONG   PostExPan;    // Level measured after Gain/Volume/Mute but without
                          Pan
  LONG   Post;         // Level measured after Gain/Pan/Volume/Mute
  BOOL   AdOvl;        // Trace PRO; Not available for Trace Alpha

  BOOL   ParChanged;   // Any of channel parameters has changed - read
                          parameter values again
  LONG   Code;         // Channel specific Error Code of last Get/Request
                          Operation
  LONG   Requests;     // Number of clients who need the level info of this
                          channel
  ULONG  Handle;       // Driver spec. Handle, do not use
  ULONG  hDev;         // Driver spec. Handle, do not use
} TALP_CHNLEVELINFO, *PTALP_CHNLEVELINFO;

typedef struct {
  LONG CardId;        // IN: ID of target card
  LONG reserved;
  TALP_CHNLEVELINFO Levels[TALP_NUMCHANNELS]; //  LevelInfo for each Mono
                                                  signal available; indexes
                                                  see: talp_lvi_XXX constants
} TALP_LEVELINFO, *PTALP_LEVELINFO;
```

The structure `TALP_LEVELINFO` with its array of `TALP_CHNLEVELINFO` is used to request and get the level values for the different mono channels. For each mono channel a `TALP_CHNLEVELINFO` structure is assigned in the `Levels` array of `TALP_LEVELINFO`. Please see the array indexes in the following table and the *"talpifc.h".*

Index	Channel
0	Analog Input 1
1	Analog Input 2
2	Digital Input L
3	Digital Input R
4	Play 1
5	Play 2
6	Play 3
7	Play 4
8	Play 5
9	Play 6
10	Play 7
11	Play 8
12	Tdm 1
13	Tdm 2
14	Tdm 3
15	Tdm 4
16	Tdm 5
17	Tdm 6
18	Tdm 7
19	Tdm 8
20	Play 9
21	Play 10
22	Play 11
23	Play 12
24	Play 13
25	Play 14
26	Play 15
27	Play 16
28	Master Sum L
29	Master Sum R
30	Aux 1
31	Aux 2
32	Aux 3
33	Aux 4
34	Aux 5
35	Aux 6

talp_RequestLevel

```
LONG TALPCALL talp_RequestLevel (PTALP_LEVELINFO pLevelInfo);
// Switchs Level Anaysis for specific channels on/off
// Input:
//   pLevelInfo = pointer to level information structure
//   pLevelInfo->CardId = ID of target card
//   pLevelInfo->Levels[].Requests = initialized with >0 if Level required
// Output:
//   pLevelInfo->Levels[].Code = Error Code for Request
//   pLevelInfo->Levels[].Handle = Driver specific handle for client channel
//   return < 0 ? Error : Success
```

This function switches the level analysis for specific channels on and off.

Prior the first call, all members of `pLevelInfo` must be set to zero. `pLevelInfo->Levels[].Requests` should be used to indicate, which channel level is required.

The client software can check the success of the operation for each channel by use of `pLevelInfo->Levels[].Code`. A value less than 0 indicates that the request could not be executed. This may occur with the Digital/TDM input channel only because of inappropriate clock settings (see users manual).

Example: The client software request the levels for the digital Input and any of the TDM inputs. With the default clock settings the driver assumes that both digital input clocks are not synchronized. Since the Trace Alpha always works with one clock only, the signal of the digital input can be evaluated by using the clock of this input as reference. In this case, the TDM input clock can not be evaluated.

The client software should also observe the information in `pLevelInfo->Levels[].Code` after calling `talp_GetLevel` because it may change because of recording/playback requests from audio recording applications. Read more about this in `talp_GetLevel`.

The client software must use always the identical `TALP_LEVELINFO` variable for `talp_RequestLevel` and `talp_GetLevel` calls because the driver writes client related information into this structure.

The client software must deactivate the level analysis for all channels before it exits.

talp_GetLevel

```
LONG TALPCALL talp_GetLevel (PTALP_LEVELINFO pLevelInfo);
// Get the Levels for all requested MONO channels
// Input:
//   pLevelInfo = pointer to level information structure,
//   pLevelInfo->CardId = ID of target card
// Output:
//   pLevelInfo->Levels[].Code       = Error Code for Get
                                       ( < 0 ? Error : Success)
//   pLevelInfo->Levels[].Pre        = Level of the Source Signal
                                       (PRE Gain/Fader)
//   pLevelInfo->Levels[].PostGain   = Level measured after Gain
//   pLevelInfo->Levels[].PostExPan  = Level measured after Gain/Volume/Mute
                                       but without Pan
//   pLevelInfo->Levels[].Post       = Level of the Signal after Gain, Pan,
                                       Fader, Mute
//   pLevelInfo->Levels[].AdOvl      = if TRUE then the AD Converter signaled
                                       an Input Overload
//   pLevelInfo->Levels[].ParChanged = if TRUE then any parameter of the
                                       channel has changed
//   return < 0 ? Error : Success
```

Using this function, the client software can get the levels of the different channels. The client software gets the level peak values that occurred between this and the previous call. A call of `talp_GetLevel` reset the driver stored peak value to zero.

Before the first call of `talp_GetLevel`, the `TALP_LEVELINFO` variable must be initialized with `talp_RequestLevel`.

The level value in `Pre, Pos`t of the `TALP_CHNLEVELINFO` structure is expressed like a PCM sample value. A conversion to a dB value can be done in the same way like the parameter values are converted (see *Mixer Channel Control*).

The following table contains the value ranges of the level values.

`pLevelInfo->Levels[].Pre`	`0..0x08000 == -INF ..  0 dB`
`pLevelInfo->Levels[].PostXXX`	`0..0x08000 == -INF .. +6 dB`

The level value in `Pre` of the `TALP_CHNLEVELINFO` structure always represent the level of the source signal. That means for

- Analog Input Channels the level after the gain control
- All other Mixer Channels the level prior the gain control
- Sum Channels the level prior the fader

The level value in `Post` of the `TALP_CHNLEVELINFO` structure always represent the level post fader. This way it is very easy for the client software to implement a switchable "pre fader metering".

The client software should use `pLevelInfo->Levels[].Code` to evaluate to validity of the level values. This code can contain an error state for that channel even if `talp_RequestLevel` not returned an error. This can be the case for the Digital and TDM input channels only.

Example: Like mentioned, the Trace Alpha works always with one reference clock only, means also with one sample rate only. If a client software requests a level information for the digital input with sample rate converters switched off, the driver uses the clock of this input. Imagine the digital input runs at 44.1 kHz and a recording software requests a recording from the analog input at 48 kHz. Since recording and playback have always the higher priority towards level analysis, the driver, according its default clock settings, would switch to the internal clock and would set it to 48 kHz. In this case, the level of the digital input could not be evaluated and the `pLevelInfo->Levels[].Code` would signal an error. After the recording has finished, the driver looks at the level request table and ensures the right clock settings for the requested levels. The `pLevelInfo->Levels[].Code` would signal no error again.