H-48928

MODEL 7680B

Versatile Scientific Sampling Processor
K5/VSSP32
Driver Software

Manual

Japan Communication Equipment Co., Ltd.

1 Introduction

The textbook describes the driver of USB device ”Time synchronous data sampler " that was programmed in order to operate under the Linux Kernel 2. 6 series.

2 Operational verification environment

- Debian GNU / Linux 5.0 (kernel 2. 6. 26 - 1 - 686) i686 platform[1][2]
- The sampling data is retained at binary type in the local hard disk.

[1] The operation is only confirmed under the software development environment “Debian GNU / Linux 5.0 (kernel 2. 6. 26 - 1 - 686)“.
[2] The operation in the multiprocessor system, or the operation if you installed the multiple target samplers in the same PC is not corresponded to.

3 Directory /File constitution

The constitution of Directory/file that thaws the archive which was created by tar + gzip is as follows.

```
vlbi-usb-linux-1.18/
    |
    + Makefile
    + doc /
    |   |
    |   + Makefile      Makefile for instruction manual compilation
    |   |
    |   + manual.pdf    Manual (PDF type)
    |   |
    |   + manual.ps     Manual (PostScript type)
    |   |
    |   + manllal.tex   Manual (LaTex source)
    |
    + driver /
    |   |
    |   + Makefile      Makefile for builds UTDS Driver
    |   |
    |   +utds.c         UTDS Driver source file
    |
    + examples /
    |   |
    |   +Makefile       Makefile for build Sample program
    |   |
    |   +*.c            Sample program（C source）
    |
    + include /
    |   |
    |   + tdsio.h       UTDS driver I / O Control command file
    |   |
    |   + tdssdh . h Header format defined file of sampling data of time synchronous type data
sampler
    |
    + libtds /
        |
        +Makefile       Makefile for build library
        |
        + 1ibtds.c      library source file
```

4 Loadable modules

4.1 Create the loadable module

(a) Decompress vlbi-usb-linux-1.18.tar.gz and move the files to vlbi-usb-linux-1.18.

```
# tar xvfz vlbi-usb-linux-1.18.tar.gz
# cd vlbi-usb-linux-1.18/
```

(b) Construct the module and install. At the same time build the library and install it.

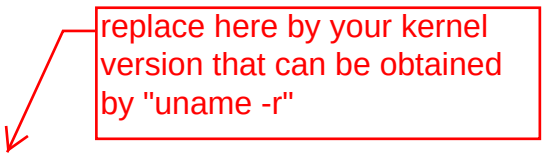
```
# make
# make install
```

(c) Create the device node.

```
# cd   /dev
# mknod -m 666 utds0 c 180 222
```

The device file becomes to (/dev/utds0). When udevd is operational on Linux, it is not necessary to create the device node.

(d) Load the module.




```
# insmod /lib/modules/2.6.26-1-686/kernel/drivers/usb/misc/utds.ko
```

5 Install files

By execute “make install”, the following files are installed to the respective place.

tds.ko / lib / modules / 2.6.26 -1 -686 / kernel / drivers / usb / misc / utds.ko

tdsio.h / usr / include / sys / tdsio.h

tdssdh.h / usr / include / sys / tdssdh.h

libtds . 50 / usr / local / lib / libtds .so

6 System call interface

6.1 Supporting system call

1. open ()
2. close ()
3. read ()
4. ioctl ()

6. 2 Using examples for system call

As follows, there shows examples of each system call. Other system calls which has not been described is according to the specification of system call of Linux Kernel 2.6 series.

1. open ()

Open the device and return the file descriptor. Device name and open mode are appointed to argument. Since this device is read-only, so O_RDONLY should be appointed to parameter “open mode”.

(Example)

```
int fd ;

fd = open ( ' " / dev / utds0 ", 0 _ RDONLY )
```

2. close ( )

Close the device. Appoint the return value of open () to file descriptor parameter

(Example)

```
close( fd );
```

3. read ( )

Read the data from the opend device. Appoint file descriptor (the return value of open ()), buffer address and buffer size for data reading. The size (bytes) of the data read will be returned.

(Example)

```
int nread ;
unsigned int buffer [ 1024 ] ;
int nbyte = sizoef ( buffer ) ;

nread = read ( fd , buffer , nbyte )
```

4. ioctl ( )

Do the various operations of device. Appoint file descriptor (the return value of open ()) to 1st argument, the operation should be done to 2nd argument and the other parameters when necessary.

- Initialization

  Initialize parameters inside device driver. TDSIO _ INIT should be appointed to 2nd argument. When this operation is executed, it becomes state as follow.

  --When the number of channels setting has not be done (C.f. 4)
  --When bits several setting have not be done (C.f. 4)
  --When filter frequency setting has not be done（C.f. 4）
  --When sampling frequency setting has not be done（C.f. 4）
  --When 1PPS Synchronization has not be done（C.f. 4）
  --When time setting has not be done（C.f. 4）
  --When error information is reseted（C.f. 4）

  (Example)

```
ioctl ( fd , TDSIO _ INIT ) ;
```

- Sampler FIFO Clearing
  Clear the FIFO on the sampler . TDSIO _ BUFFER _ CLEAR should be appointed to 2nd argument.

  (Example)

  ioctl (fd , TDSIO _ BUFFER _ CLEAR) ;

- 1PPS Synchronization
  Do the Synchronization of the external 1PPS input signal.
  TDSIO_SYNC_IPPS should be appointed to 2nd argument. When it cannot detect the external 1PPS input signal, error occurs.

  (Example)

  ioctl (fd , TDSIO _ SYNC _ IPPS) ;

- Time setting.
  Set the time to the sampler. TDSIO _ SET _ TIME should be appointed to 2nd argument, and the address of the variable which keeps the time data (32bit) should be appointed to 3rd argument.
  The type of time appoints subordinate 17 bits (obit - 16bit) at total second from 00: 00: 00 and 18bit - 27bit (9bit width) at total days from January 1st.[3]

  A simple macro is prepared below.

Macro	Function
TDS_MAKE_TIME (day, sec)	With day (total days from January 1st) and sec (Total second from 00: 00: 00), appropriate value will be created.

  (Example)

  / * When 2000/09/22 14: 39: 28 is appointed */
  unsigned int time ;
  unsigned int day

```
unsigned int sec

day = 31 + 29 + 31 + 30 + 31 + 30 + 31 + 31 + 22;
sec = ( ( 14 * 60)＋39 ) * 60 + 28;
time = TDS_MAKE_TIME ( day , sec )

ioctl ( fd , TDSIO_SET_TIME , & time ) ;
```

[3]The same type of the time that is appointed when writing in to the device register

- Time acquisition.

  Acquire the time that the sampler keeps. TDSIO_GET_TIME should be appointed to 2nd argument, and the address(32bit) of the variable which houses the time data that is acquired should be appointed to 3rd argument. As for type of the time data that is acquired, similar to the type in time setting.

  A simple macro is prepared below.

Macro	Function
TDS_GET _SEC (arg)	Get the total seconds from 00: 00: 00, from the argument arg
TDS_GET _SEC_CARRY (arg)	Get the carry bit of the number of total seconds from the argument arg. The result will be 0 or 1.
TDS_GET_DAYS (arg)	Get the Total days from January 1st from the argument arg.
TDS_GET_DAYS_CARBY (arg)	Get the carry bit of the number of total days from the argument arg. The result will be 0 or 1.

(Example)

```
unsigned int time ;
unsigned int day;
unsigned int sec;

ioctl ( fd , TDSIO_GET_TIME , & time ) ;

day = TDS_GET_DAYS ( time ) ;
sec = TDS_GET_SEC ( time ) ;
```

- Year information setting

  Set the year information to the sampler. TDSIO_GET_YEAR should be appointed to 2nd argument, and the address(32bit) of the variable that houses the year data should be appointed to 3rd argument.

(Example)

```
unsigned int year ;
```

ioctl (fd , TDSIO _ SET _ YEAR , & year)

- Year information acquisition.
  Get the year information that the sampler keeps. TDSIO_ GET_YEAR should be appointed to 2nd argument, and the address(32bit) of the variable that houses the year data should be appointed to 3rd argument.

  (Example)

  unsigned int year ;

  ioctl (fd , TDSIO_GET_YEAR , & year)

- Channel several setting.
  Set the number of channels that do sampling. TDSIO_SET_CHMODE should be appointed to 2nd argument, and the address (32bit) of the variable that keeps the information of channel several setting should be appointed to 3rd argument.

  --Channel several appointments

The number of channels	Designated macro
1ch	TDSIO_SAMPLING_1CH
4ch	TDSIO_SAMPLING_4CH

  (Example)

  / * When 4ch is set*/
  unsigned int chmode;

  chmode = TDS IO_SAMPLING_4CH ;

  ioctl (fd , TDSIO_SET_CHMODE , & chmode) ;

- Bit number setting.
  Set the bit number that does sampling. TDSIO_SET_RESOLUTIONBIT should be appointed to 2nd argument , and the address(32bit) of the variable that keeps the information of bit number setting should be appointed to 3rd argument.

  -- Data width appointments

Bit number	Designated macro
1bit	TDSIO_SAMPLING_1BIT
2bit	TDSIO_SAMPLING_2BIT
4bit	TDSIO_SAMPLING_4BIT
8bit	TDSIO_SAMPLING_8BIT

  (Example)

```
/ * When 1bit is set */
unsigned int resolutionbit ;

resolutionbit = TDSIO_SAMPLING_1BIT ;

ioctl ( fd , TDSIO_SET_RESOLUTIONBIT ,& resolutionbit);
```

- Filter frequency setting.
  Set the filter frequency that does sampling. TDSIO_SET_FILTER should be appointed to 2nd argument, and the address(32bit) of the variable that keeps the information of filter frequency setting should be appointed to 3rd argument.

  -- Filter frequency appointments

Filter frequency assignment	Designated macro
16M	TDSIO_SAMPLING_16M
8M	TDSIO_SAMPLING_8M
4M	TDSIO_SAMPLING_4M
2M	TDSIO_SAMPLING_2M
Thru	TDSIO_SAMPLING_THRU

  (Example)

```
/ * When Thru is set */
unsigned int filter ;
```

filter = TDSIO_SAMPLING_THRU ;

ioctl (fd , TDSIO _ SET _ FILTER , & filter) ;

- Sampling setting.
  Set the sampling frequency that does sampling. TDSIO_SET_FSAMPLE should be appointed to 2nd argument, and the address(32bit) of the variable that keeps the information of sampling frequency setting should be appointed to 3rd argument.

-- Frequency assignment

Sampling frequency	Designated macro
40kHz	TDSIO_SAMPLING_40KHZ
100kHz	TDSIO_SAMPLING_100KHZ
200kHz	TDSIO_SAMPLING_200KHZ
500kHz	TDSIO_SAMPLING_500KHZ
1MHz	TDSIO_SAMPLING_1MHZ
2MHz	TDSIO_SAMPLING_2MHZ
4MHz	TDSIO_SAMPLING_4MHZ
8MHz	TDSIO_SAMPLING_8MHZ
16MHz	TDSIO_SAMPLING_16MHZ

(Example)

/ * When 1MHz is set */
unsigned int fsample ;

fsample = TDSIO_SAMPLING_1MHZ ;

ioctl (fd , TDSIO _ SET _ FSAMPLE , & fsample)

- Channel several setting acquisitions
  Get the present channel several setting. TDSIO_GET_CHMODE should be appointed to 2nd argument, and the address(32bit) of the variable that keeps the information of the setting that is acquired should be appointed to 3rd argument. Information of channel several setting which are obtained will be the same type as those which are appointed at the time of channel several setting of 4.

  (Example)

```
unsigned int chmode ;

ioctl ( fd , TDSIO_GET_CHMODE , & chmode ) ;

if ( chmode = = TDSIO_SAMPLING_4CH ) {
/ * 4CH * /
}
```

- Bit number setting acquisition
  Get the present bit number setting. TDSIO_GET_RESOLUTIONBIT should be appointed to 2nd argument, and the address(32bit) of the variable that keeps the information of the setting that is acquired should be appointed to 3rd argument. Information of the bit number setting that is obtained will be the same type as those that are appointed at the time of bit number setting of 4.

  (Example)

```
unsigned int resolutionbit

ioctl ( fd , TDSIO _ GET _ RESOLUTIONBIT , &resolutionbit ) ;

if ( resolutionbit = = TDSIO_SAMPLING_4BIT ) {
/ * 4Bit * /
}
```

- Filter frequency setting acquisition
  Get the present filter frequency setting. TDSIO_GET_FILTER should be appointed to 2nd argument, and the address (32bit) of the variable that keeps the information of the setting that is acquired should be appointed to 3rd argument. Information of the filter frequency setting that is obtained will be the same type as those that are appointed at the time of filter frequency setting of 4.

  (Example)

```
unsigned int filter ;

ioctl ( fd , TDSIO_GET_FILTER , & filter )

if ( filter = = TDSIO _ SAMPLING _ THRU ) {
 / * Thru * /
 }
```

- Sampling frequency setting acquisition
  Get the present sampling frequency setting. TDSIO_GET_FSAMPLE should be set to 2nd argument, and the address(32bit) of the variable that keeps the information of the setting that is acquired should be appointed to 3rd argument. Information of the sampling frequency setting that is obtained will be the same type as those that are appointed at the time of sampling frequency setting of 4.

  (Example)

```
unsigned int fsample ;

ioctl ( fd , TDSIO_GET_FSAMPLE, & fsample ) ;

if (fsample = = TDSIO _ SAMPLING _ 4MHZ ) {
 / * 4MHz * /
 }
```

- Acquisition of status

  Get the status information of the sampler. TDSIO_GET_STATUS should be set to 2nd argument, and the address(32bit) of the variable that keeps the status information should be appointed to 3rd argument.[4]

  (Example)

```
unsigned int stat ;

ioctl ( fd , TDSIO_GET_STATUS, & stat ) ;
```

- Start of sampling

  Indication to start the sampling . TDSIO_SAMPLING_START should be appointed to 2nd argument.

  (Example)

```
ioctl ( fd , TDSIO_SAMPLING_START) ;
```

[4] The status information that is obtained reaches the value that is obtained from the register the sampler has.

- Sampling stop

  Indication to stop the sampling. TDSIO_SAMPLING_STOP should be appointed to 2nd argument.

  Furthermore, when all of the process which has opened the device (open()) is terminated on host PC, sampling will be stopped automatically.

  (Example)

  ```
  ioctl ( fd , TDSIO_SAMPLING_STOP) ;
  ```

- DC offset setting

  Set the present DC offset value. TDSIO_SET_DCOFFSET should be appointed to 2nd argument, and the address of the structure (struct tdsio) that houses the setting information should be appointed to 3rd argument.

  If you want to get the value of DC offset (1CH), 0 should be set to the member key of struct tdsio, else if you want to get the value of 2CH, 1 should be set.

  (Example)

  ```
  struct tdsio tdsio ;

  tdsio.key = 0 ; / * DCoffset 1 * /
  tdsio.value = 0x10 ;
  ioctl ( fd , TDSIO_SET_DCOFFSET, & tdsio ) ;
  ```

  Dc offset acquisition

  Get the present DC offset value setting. TDSIO_GET_DCOFFSET should be appointed to 2nd argument, and the address of the structure (struct tdsio) that houses the setting information should be appointed to 3rd argument.

  If you want to get the value of DC offset (1CH), 0 should be set to the member key of struct tdsio, else if you want to get the value of 2CH, 1 should be set.

(Example)

```
struct tdsio tdsio ;

tdsio.key= 0 ; / * DCoffset 1 * /
ioctl ( fd , TDSIO_GET_DCOFFSET, & tdsio )

tdsio.key= 1 ; / * DCoffset 2 * /
ioctl ( fd , TDSIO_GET_DCOFFSET, & tdsio )
```

- Sampler FIFO reset

  Reset the FIFO on the sampler. TDSIO_BUFFER_RESET should be appointed to 2nd argument.

(Example)

```
ioctl ( fd , TDSIO_BUFFER_RESET) ;
```

- 1PPS input status check

  Check the 1PPS input state to the sampler. TDSIO_GET_1PPSINPUT should be appointed to 2nd argument.

(Example)

```
unsigned int pps

ioctl ( fd , TDSIO _ GET _ IPPSINPUT , & pps )
if ( pps = = 0 {
/ * no 1PPS Input * /
}
```

- Reference signal input status check

  Check the reference signal input state to the sampler.
  TDSIO_GET_REFINPUT should be appointed to 2nd argument.

  (Example)

```
unsigned int ref ;

ioctl ( fd , TDSIO_GET_REFINPUT, & ref ) ;
if ( ref ==0) {
/* no Ref Input * /
}
```

- 5 / 10MHz Input signal decision

  It checks whether the reference signal input to the sampler is 5MHz or 10MHz.In case of 5MHz 1, in case of 10MHz 0 will be set.
  TDSIO_GET_10MHZINPUT should be appointed to 2nd argument.

  (Example)

```
unsigned int rhz ;

ioctl ( fd , TDSIO _ GET _ 10MHZINPUT , & rhz ) ;
 if ( rhz = = 0 ) {
 / * 10MHz * /
 }
```

- AUX Data setting

  Set the AUX data .TDSIO_SET_AUX should be appointed to 2nd argument, and the address of the structure (struct tdsauxio) that houses the AUX data should be appointed to 3rd argument.

The size of AUX data will be set to member auxsize, and The AUX data will be set to member auxfield. The maximum data size that can be set to auxfidd is 256 bytes.

(Example)

```
struct tdsauxio tdsauxio ;

tdsauxio . auxsize = 10 ;
memcpy ( tdsio . auxfield , " TEST AUX ! " , tdsauxio . auxsize )

ioctl ( fd , TDSIO_SET_AUX, & tdsauxio ) ;
```

- AUX Data acquisition

  Get the AUX data. TDSIO_GET_AUX should be appointed to 2nd argument, and the address of the structure (struct tdsauxio) that houses the AUX data should be appointed to 3rd argument. After the acquiring, you can get the size of the AUX data from member auxsize and you can also get the AUX data from member auxfield.

  (Example)

```
struct tdsauxio tdsauxio ;
char buf [ 256 ] ;

ioctl ( fd , TDSIO_GET_AUX, & tdsauxio ) ;

memcpy ( buf , tdsio . auxfield , tdsauxio . auxsize ) ;
```

- Error information acquisition

  Get the information of the error that occurs when operate the device. TDSIO_GET_ERROR should be appointed to 2nd argument, and the address(32bit) of the variable which houses the error information should be appointed to 3rd argument.

Immediately after host PC boots , or when 4 initialization is executed, error information will be reset. After that,When error occurs, the information of that error type will be kept. When furthermore errors occur, the information will be overwrited. Therefore,the value you get by Error information acquisition only indicates the type of the error that occurs most recently.

Error classification has defined below.[5]

[5] vlbi - usb - linux / include / tdsio.h reference

Error value	Error classification
0	No error.(Normal)
1	The error is not applicable to the classification below.
2	Operation or the parameter not supporting was appointed.
3	The operation that should be done after sampling started was appointed when sampling has not been started.
4	The operation that should be done before sampling start was appointed after sampling has been started.
5	The operation that should be done when sampling setting (frequency/data width/the number of channels) is done was appointed when setting is not be done.
6	FIFO on the board overflowed.
7	The external 1PPS input signal can not be detected.
8	The external 10MHz input signal can not be detected.
9	The external time input signal can not be detected.
10	The operation that should be done after synchronization of 1PPS was appointed when 1PPS is not synchronized
11	The operation that should be done after time synchronization or time setting was appointed before time synchronization or time setting.
12	Try to execute read () ,when it is executed by other process.
13	The error when the extended board for the PCI board is not connected.
14	The DMA buffer is not enough.
15	Error when I/O processing with the user land.
16	DMA processing error.
17	No response from the data sampler during fixed time.
18	Detected the data that has not aligned at the word unit
19	The header was not found in the position where you suppose inside the sampling data.
20	The header and the recognition possible data existed even in the rubbish data when starting the sampling.

(Example)

```
int err ;

ioctl ( fd , TDSIO _ GET _ ERROR , & err );
```

- The register reading

  It reads the value of the optional register. TDSIO_DEGIO_READ should be appointed to 2nd argument, and the address of the structure (struct tdsio _regio) that houses the value of the register should be appointed to 3rd argument.

  After reading out, register address will be housed in member address,present value will be housed in member value.

  (Example)

  struct tdsio _ regio regio

  ioctl (fd , TDSIO _ REGIO _ READ , & regio) ;
  printf (“address = ox % 02x , value = ox % 02x ¥ n“ , regio.address ,
  regio.value)

- Version information reading,

  It reads out the version information value of the driver.
  TDSIO_GET_VERSION should be appointed to 2nd argument, and the address of the variable which houses the version information should be appointed to 3rd argument. Version information consists of three values ,the major, the minor, and the revision number. To get the respective number, use the following macro.

TDS _VERSION_MAJOR ()	The macro which gets the MAJOR number
TDS _VERSION_MINOR ()	The macro which gets the MINOR number
TDS_VERSION_REVISION ()	The macro which gets the REVISION number

  (Example)

  uint32 _ t version ;

  ioctl (fd , TDSIO _ GET _ VERSION , & version) ;

```
printf ( " version = % 02d . % 02d . % 02d ¥ n ", TDS_VERSION_MAJDR
    (version), TDS _ VERSION _ MINOR (version), TDS _ VERSION _
    REVISION (version));
```

- FPGA version information reading,
  It reads out the FPGA version information value of the driver. TDSIO_GET_VERFPGA should be appointed to 2nd argument. The most significant 2-bit of returned data indicates major version number, the least significant 6-bit of returned data indicates major version number

  (Example)

```
unsigned int fpgaver ;
unsigned char major, minor ;
ioctl ( fd , TDSIO _ GET _ VERFPGA , & fpgaver ) ;

major   = fpgaver >> 6 ;
minor   = fpgaver & 0x3f ;
```

- Bit shift information setting
  Sets the bit shift information to the sampler. TDSIO_SET_BITSHIFT should be appointed to 2nd argument.

  (Example)

```
/ * When bit shift is 2 */
unsigned int bitshift ;

bitshift = 2 ;
ioctl ( fd , TDSIO _ SET _ BITSHIFT , & bitshift )
```

- Bit shift information acquisition.
  Gets the bit shift information from the sampler. TDSIO_GET_BITSHIFT should be appointed to 2nd argument.

(Example)

```
unsigned int bitshift ;
ioctl ( fd , TDSIO_GET_YEAR , & bitshift )

if   (bitshift == 0) {
/ * Bit Shift Setting is 0 */
}
```

7 Sample program

Please refer to example.c as a sample program below Vlbi - usb - linux/sample.

8 Library

The library to abstract the system call is created. The library will be built, installed with driver module.

8. 1 Notice when using

When the library is been used, it is necessary to make like below.

- Include < sys/tdsio.h >.
  In source file, describe as this
  # include < sys / tdsio.h >

- Link libtds.
  When linking -ltds is appointed to option.

8.2 Use of library function

Usage of each library function are as follows. The UTDS driver does not support the function of part as for which is left as compatibility to the TDS drivers for 9260 data sampler.

1. TdsOpen ()

Open the device.

Argument 1	char *(Character)	Device name is appointed. When NULL is appointed, with default " Dev/tds0 " is used.
Return value	tdsdev_t *	As for details of this variable, it is not necessary to have been concerned on user side. When it fails in opening, NULL is returned. Below, it utilizes this return value, with all library functions as argument.

2. TdsClose ()

Close the device.

Argument 1	tdsdev_t *	The return value of TdsOpen() of 1 should be appointed
Return value	int	When it succeeds, 0 is returned.

3. TdsInit ()

Initialize the variable inside the driver

Argument 1	tdsdev_t *	The return value of TdsOpen() of 1 should be appointed
Return value	int	When it succeeds, 0 is returned. When it fails, value other than 0 is returned

4.TdsClrBuffer ()

Initialize FIFO on the sampler

Argument 1	tdsdev_t *	The return value of TdsOpen() of 1 should be appointed
Return value	int	When it succeeds, 0 is returned. When it fails, value other than 0 is returned

5. TdsSync1pps ()

Synchronize external 1PPS input signal

Argument 1	tdsdev_t *	The return value of TdsOpen() of 1 should be appointed
Return value	int	When it succeeds, 0 is returned. When it fails, value other than 0 is returned

6. TdsSyncTime ()

Synchronize external time input signal

Argument 1	tdsdev_t *	The return value of TdsOpen() of 1 should be appointed
Return value	int	When it succeeds, 0 is returned. When it fails, value other than 0 is returned

7. TdsSetTime ( )

Set time to the sampler.

Argument 1	tdsdev_t *	The return value of TdsOpen() of 1 should be appointed
Argument 2	tds _time_t	Appointing the address of the variable that keeps time. When NULL is appointed,use the time get from OS .
Return value	int	When it succeeds, 0 is returned. When it fails, value other than 0 is returned

The definition of tds_time_t is as follows.

```
struct   tds _time {
        int sec ;
        int sec_carry ;
        int day ;
        int day _carry ;
};
typedef struct tds_time tds_time_t ;
```

sec	Total seconds From 00: 00: 00
sec_carry	Carry of second. If there is carry, 1, or not, 0. but in case of time setting, value of this field is ignored.
day	Total days from January 1st
day_carry	Carry of day. If there is carry, 1, or not, 0. but in case of time setting, value of this field is ignored.

8. TdsGetTime ( )

Get the time which the sampler keeps .

Argument 1	tdsdev_t *	The return value of TdsOpen() of 1 should be appointed
Argument 2	tds _time_t *	Address of the variable that houses time is appointed.
Return value	int	When it succeeds, 0 is returned. When it fails, value other than 0 is returned

About the type definition of (tds _ time _t), referring to 7.

9. TdsSetYear ()

Set year information to the sampler.

Argument 1	tdsdev_t *	The return value of TdsOpen() of 1 should be appointed
Argument 2	unsigned int *	Address of the variable which houses year information is appointed
Return value	int	When it succeeds, 0 is returned. When it fails, value other than 0 is returned

10. TdsGetYear ()

Get the year information which the sampler keeps.

Argument 1	tdsdev_t *	The return value of TdsOpen() of 1 should be appointed
Argument 2	unsigned int *	Address of the variable which houses year information
Return value	int	When it succeeds, 0 is returned. When it fails, value other than 0 is returned

11. TdsSetSamPling () [6]

Do Sampling setting

Argument 1	tdsdev_t *	The return value of TdsOpen() of 1 should be appointed
Argument 2	tds _samp_t *	Address of the variable that keeps sampling setting.
Return value	int	When it succeeds, 0 is returned. When it fails, value other than 0 is returned

Type definition of tds_samp_t is as follows.

```
struct tds_samp {
        int freq;
        int unit;   /*'k' or 'M'*/
        int width ;
        int channel ;
 };

typedef struct tds_ samp tds_samp_t ;
```

[6] MODEL 9270, 7680B does not support

<table>
<tr><td>freq,
unit</td><td>Appointe sampling frequency is
<table><tr><th>Frequency</th><th>freq</th><th>Unit</th></tr><tr><td>40kHz</td><td>40</td><td>’k’</td></tr><tr><td>100kHz</td><td>100</td><td>’k’</td></tr><tr><td>200kHz</td><td>200</td><td>’k’</td></tr><tr><td>500kHz</td><td>500</td><td>’k’</td></tr><tr><td>1MHz</td><td>1</td><td>’M’</td></tr><tr><td>2 MHz</td><td>2</td><td>’M’</td></tr><tr><td>4 MHz</td><td>4</td><td>’M’</td></tr><tr><td>8 MHz</td><td>8</td><td>’M’</td></tr><tr><td>16 MHz</td><td>16</td><td>’M’</td></tr></table></td></tr>
<tr><td>width</td><td>Appointe sampling bit width.
<table><tr><th>Bit width</th><th>Width</th></tr><tr><td>1</td><td>1</td></tr><tr><td>2</td><td>2</td></tr><tr><td>4</td><td>4</td></tr><tr><td>8</td><td>8</td></tr></table></td></tr>
<tr><td>channel</td><td>Appointe the number of sampling channels.
<table><tr><th>The number of channels</th><th>Channel</th></tr><tr><td>1</td><td>1</td></tr><tr><td>4</td><td>4</td></tr></table></td></tr>
</table>

12. TdsGetSampling () [7]

Get the information of sampling setting of the sampler.

Argument 1	tdsdev_t *	The return value of TdsOpen() of 1 should be appointed
Argument 2	tds_samp_t*	Address of the variable which houses sampling setting information
Return value	int	When it succeeds, 0 is returned. When it fails, value other than 0 is returned

Concerning the definition of tds _samp_ t, referring to 11

[7] MODEL 9270, 7680B does not support

13. TdsSetSampling2()

Do sampling setting

Argument 1	tdsdev _ t *	The return value of TdsOpen() of 1 should be appointed
Argument 2	int *	Address of the variable which houses sampling frequency specification
Argument 3	int *	Address of the variable which houses filter frequency specification
Argument 4	int *	Address of the variable which houses bit number specification
Argument 5	int *	Address of the variable which houses channel several specification.
Return value	int	When it succeeds, 0 is returned. When it fails, value other than 0 is returned

14. TdsGetSampling2 ()

Get the information of sampling setting of the sampler.

Argument 1	tdsdev _ t *	The return value of TdsOpen() of 1 should be appointed
Argument 2	int *	Address of the variable which sampling frequency specification should be set
Argument 3	int *	Address of the variable which filter frequency specification should be set
Argument 4	int *	Address of the variable which bit number specification should be set
Argument 5	int *	Address of the variable which channel several specification should be set
Return value	int	When it succeeds, 0 is returned. When it fails, value other than 0 is returned

15. TdsSetDcoffset ()

Set the information of DC offset value to the sampler.

Argument 1	tdsdev_t *	The return value of TdsOpen() of 1 should be appointed
Argument 2	int *	Address of the variable that keeps the number of the channel that should set DC offset value should be appointed
Argument 3	unsigned int *	Address of the variable which keeps the DC offset value
Return value	int	When it succeeds, 0 is returned. When it fails, value other than 0 is returned

16. TdsGetDcoffset ()

Get the information of DC offset value from the sampler.

Argument 1	tdsdev_ t *	The return value of TdsOpen() of 1 should be appointed
Argument 2	int *	Address of the variable that keeps the number of the channel of the DC-offset value which you get should be appointed.
Argument 3	unsigned int *	Address of the variable which houses DC offset value should be appointed.
Return value	int	When it succeeds, 0 is returned. When it fails, value other than 0 is returned

17. TdsStatus ()

Get the status information that the sampler keeps.

Argument 1	tdsdev_t *	The return value of TdsOpen() of 1 should be appointed
Argument 2	unsigned int *	Address of the variable that should house status information should be appointed. The status information obtained is register value itself of the sampler.
Return value	int	When it succeeds, 0 is returned. When it fails, value other than 0 is returned

18. TdsStart ()

Start the sampling

Argument 1	tdsdev_t *	The return value of TdsOpen() of 1 should be appointed
Return value	int	When it succeeds, 0 is returned. When it fails, value other than 0 is returned

19. TdsStop ()

Stop the sampling

Argument 1	tdsdev_t *	The return value of TdsOpen() of 1 should be appointed
Return value	int	When it succeeds, 0 is returned. When it fails, value other than 0 is returned

20. TdsGetData ()

Read double word (4 bytes) data from the sampler.

Argument 1	tdsdev_t *	The return value of TdsOpen() of 1 should be appointed
Argument 2	unsigned int *	Address of the variable which should house the data should be appointed.
Return value	int	When it succeeds, 0 is returned. When it fails, value other than 0 is returned

21. TdsRead ()

Read the appointed number of data from the sampler.

Argument 1	tdsdev_t *	The return value of TdsOpen() of 1 should be appointed
Argument 2	int *	Address of the variable that keeps the number of bytes of data that it should read. The number of bytes of data that is really read will be set in this address.
Argument 3	unsigned char *	The first address of the area (Array and etc.) where it should house the data should be appointed.
Return value	int	When it succeeds, 0 is returned. When it fails, value other than 0 is returned

22. TdsSetAux ()

Set the AUX data that will be added to the header of the sampling data.

Argument 1	tdsdev_t *	The return value of TdsOpen() of 1 should be appointed
Argument 2	int *	Address of the variable that keeps the number of bytes of AUX data.
Argument 3	char *	The first address of the area (Array and etc.) where the AUX data should be set.
Return value	int	When it succeeds, 0 is returned. When it fails, value other than 0 is returned

23. TdsSetAuxFile ()

The AUX data that is read from the file is appointed to be set and it sets as the AUX data that is added to the header of the sampling data.

Argument 1	tdsdev_t *	The return value of TdsOpen() of 1 should be appointed
Argument 2	char *	The first address of the area (Array and etc.). where the path of the file that it reads as the AUX data is kept.
Return value	int	When it succeeds, 0 is returned. When it fails, value other than 0 is returned

24. TdsGetAux ()

Get the AUX data that is added to the header of the sampling data.

Argument 1	tdsdev_t *	The return value of TdsOpen() of 1 should be appointed
Argument 2	Int *	Address of the variable that houses the number of bytes of AUX data
Argument 3	char *	The first address of the area (Array and etc.) where the AUX data has been set
Return value	int	When it succeeds, 0 is returned. When it fails, value other than 0 is returned

25. TdsGetAuxFile ()

Write Aux data to the appointed file.

Argument 1	tdsdev_t *	The return value of TdsOpen() of 1 should be appointed
Argument 2	char *	The first address of the area (Array and etc.).where the path of the file (which the AUX data should be written) is kept.
Return value	int	When it succeeds, 0 is returned. When it fails, value other than 0 is returned

26. TdsError ()

Get the error information

Argument 1	tdsdev_t *	The return value of TdsOpen() of 1 should be appointed
Argument 2	int *	Address of the variable that should house the information of the error that you get.
Argument 3	char *	Address of the variable that should house the message of the error
Return value	int	When it succeeds, 0 is returned. When it fails, value other than 0 is returned

27. TdsGetVersion ()

Get the version information value of the driver and the library.

Argument 1	tdsdev _ t *	The return value of TdsOpen() of 1 should be appointed
Argument 2	int *	Address of the variable that should house the version information of the driver
Argument 3	int *	Address of the variable that should house the version information of the library.
Return value	int	When it succeeds, 0 is returned. When it fails, value other than 0 is returned

Version information consists of three values, the major, the minor and the revision number. To get the respective number, use the following macro.

TDS_VERSION_MAJOR ()	The macro which gets the major number
TDS_VERSION MINOR ()	The macro which gets the minor number
TDS_VERSION_REVISION ()	The macro which gets the revision number

8. 3

Sample program for using library functions

Please refer to libcall.c as a sample program below vlbi-usb-linux-1.18 /example.